CICE Documentation

CICE Consortium

May 03, 2024

CONTENTS

CHAPTER

ONE

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/main/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 Please do not publish results based on these data sets.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

Clone the model from the CICE-Consortium repository:

```
git clone --recurse-submodules https://github.com/CICE-Consortium/CICE
```

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

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

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

Porting information can be found in the *Porting* section. A special porting section for personal computers is in the *Porting to Laptops or Personal Computers* 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. If you need to port, see the *Porting* section as noted above. *Scripts* provides 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

Each individual release has its own Digital Object Identifier (DOI), e.g. CICE v6.1.2 has DOI 10.5281/zenodo.3888653. All versions of this lineage (e.g. CICE6) can be cited by using the DOI 10.5281/zenodo.1205674 (https://zenodo.org/record/1205674). This DOI represents all v6 releases, and will always resolve to the latest one. More information can be found by following the DOI link to zenodo.

If you use CICE, please cite the version number of the code you are using or modifying.

If using code from the CICE-Consortium repository `main` branch that includes modifications that have not yet been released with a version number, then in addition to the most recent version number, the hash at time of download can be cited, determined by executing the command `git log` in your clone.

A hash can also be cited for your own modifications, once they have been committed to a repository branch.

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

1.5 Copyright

CHAPTER

TWO

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 [27] 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,

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

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, σ 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 [51].

The downwelling longwave formula of [44] is also available in function *longwave_parkinson_washington*:

$$F_{lw\downarrow} = \epsilon\sigma T_a^4(1 - 0.261\exp\left(-7.77\times10^{-4}T_a^2\right)\left(1 + 0.275 f_{cld}\right) \tag{2.2}$$

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:

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

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, θ is the turning angle between geostrophic and surface currents, c_w is the ocean drag coefficient, ρ_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.

In addition to an ice thickness distribution, CICE includes an optional capability for a floe size distribution.

Ice floe horizontal size may change through vertical and lateral growth and melting of existing floes, freezing of new ice, wave breaking, and welding of floes in freezing conditions. The floe size distribution (FSD) is a probability function that characterizes this variability. The scheme is based on the theoretical framework described in [19] for a joint floe size and thickness distribution (FSTD), and was implemented by [47] and [48]. The joint floe size distribution is carried as an area-weighted tracer, defined as the fraction of ice belonging to a given thickness category with lateral floe size belong to a given floe size class. This development includes interactions between sea ice and ocean surface waves. Input data on ocean surface wave spectra at a single time is provided for testing, but as with the other CICE datasets, it should not be used for production runs or publications. It is not recommended to use the FSD without ocean surface waves.

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

The fundamental equation solved by CICE is [59]:

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

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, ψ 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.

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 [10] as adapted for sea ice by [38]; this scheme is discussed in Section *Horizontal Transport*. Ice is transported in thickness space using the remapping scheme of [37]. The mechanical redistribution scheme, based on [59], [52], [16], [12], and [39] 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 [21], as modified by [8], [20], [22] and [23], or a new elastic-anisotropic-plastic model [65][63][60] 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

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

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

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

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_lvl	3	aice	nt_apnd	
		apnd	nt_vpnd	

continues on next page

Table 1 – continued from previous page

flag	num tracers	dependency	index (CICE grid)	index (bio grid)
		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_iso	n_iso	vice, vsno	nt_iso	
tr_brine		vice	nt_fbri	
tr_fsd	nfsd	aice	nt_fsd	
tr_snow	nslyr	vsno	nt_rsnw	
	nslyr	vsno	nt_rhos	
	nslyr	vsno	nt_smice	
	nslyr	vsno	nt_smliq	
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, the floe size distribution, advanced snow physics, age of the ice, aerosols, water isotopes, 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 [10] as modified for sea ice by [38].

- The upwind scheme uses velocity points at the East and North face (i.e. $uvelE = u$ at the E point and $vvelN = v$ at the N point) of a T gridcell. As such, the prognostic C grid velocity components ($uvelE$ and $vvelN$) can be passed directly to the upwind transport scheme. If the upwind scheme is used with the B grid, the B grid velocities, $uvelU$ and $vvelU$ (respectively u and v at the U point) are interpolated to the E and N points first. (Note however that the upwind scheme does not transport all potentially available tracers.)
- Remapping is naturally a B-grid transport scheme as the corner (U point) velocity components $uvelU$ and $vvelU$ are used to calculate departure points. Nevertheless, the remapping scheme can also be used with the C grid by first interpolating $uvelE$ and $vvelN$ to the U points.

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 ρ in [10]. 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

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

where $\langle \nabla a \rangle$ is a centered estimate of the area gradient within the cell, α_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 [10], but there is no analog in [10] 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

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

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

where α_h and α_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

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

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

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 ϕ 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 ϕ 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 [10]. 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.

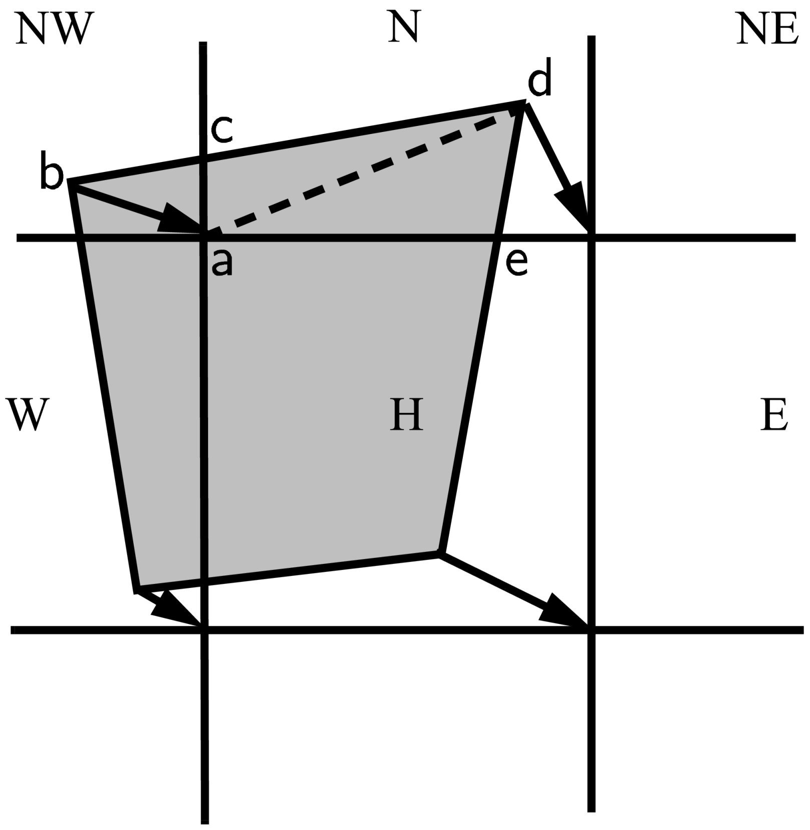


Fig. 1: Departure Region

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 [10], 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.

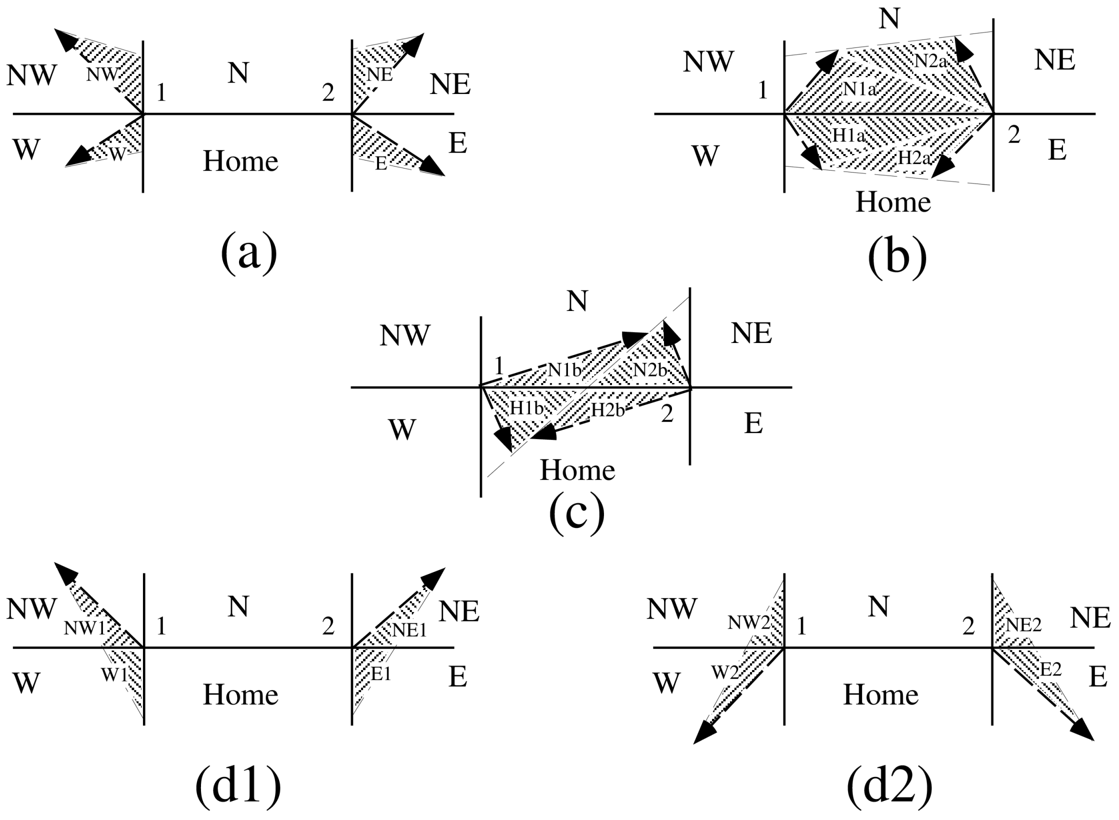


Fig. 2: Triangles

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

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

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

Given a 2D velocity field $\mathbf{u}$, 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 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 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.

The default value for the B grid is *l_fixed_area* = false. However, idealized tests with the C grid have shown that prognostic fields such as sea ice concentration exhibit a checkerboard pattern with *l_fixed_area* = false. The logical *l_fixed_area* is therefore set to true when using the C grid. The edge areas *edgearea_e* and *edgearea_n* are in this case calculated with the C grid velocity components $uvelE$ and $vvelN$.

We made one other change in the scheme of [10] 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. [10] use this formula to compute transports of the product ρT, 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 [56]:

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

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

The force balance per unit area in the ice pack is given by a two-dimensional momentum equation [15], 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 [34]. The mechanical properties of the ice are represented by the internal stress tensor σ_{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 [23] and [8].

For clarity, the two components of Equation (2.28) are

$$
\begin{aligned}
m\frac{\partial u}{\partial t} &= \frac{\partial \sigma_{1j}}{\partial x_j} + \tau_{ax} + a_i c_w \rho_w \left|\mathbf{U}_w - \mathbf{u}\right| \left[(U_w - u)\cos\theta - (V_w - v)\sin\theta\right] - C_b u + mfv - mg\frac{\partial H_\circ}{\partial x}, \\
m\frac{\partial v}{\partial t} &= \frac{\partial \sigma_{2j}}{\partial x_j} + \tau_{ay} + a_i c_w \rho_w \left|\mathbf{U}_w - \mathbf{u}\right| \left[(U_w - u)\sin\theta + (V_w - v)\cos\theta\right] - C_b v - mfu - mg\frac{\partial H_\circ}{\partial y}.
\end{aligned}
\tag{2.29}
$$

On the B grid, the equations above are solved at the U point for the collocated u and v components (see figure *Schematic of CICE B-grid.*). On the C grid, however, the two components are not collocated: the u component is at the E point while the v component is at the N point.

The B grid spatial discretization is based on a variational method described in [21] and [22]. A bilinear discretization is used for the stress terms $\partial\sigma_{ij}/\partial x_j$, which enables 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 [22]

On the C grid, however, a finite difference approach is used for the spatial discretization. The C grid discretization is based on [7], [6] and [29].

There are different approaches in the CICE code for representing sea ice rheology and for solving the sea ice momentum equation: the viscous-plastic (VP) rheology [15] with an implicit method, the elastic-viscous-plastic (EVP) [21] model which represents a modification of the VP model, the revised EVP (rEVP) approach [35][6] and the elastic-anisotropic-plastic (EAP) model which explicitly accounts for the sub-continuum anisotropy of the sea ice cover [65][63]. If `kdyn` = 1 in the namelist then the EVP model is used (module **ice_dyn_evp.F90**), while `kdyn` = 2 is associated with the EAP model (**ice_dyn_eap.F90**), and `kdyn` = 3 is associated with the VP model (**ice_dyn_vp.F90**). The rEVP approach can be used by setting `kdyn` = 1 and `revised_evp` = true in the namelist.

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 [65][60]. 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 rEVP is also a fully explicit scheme which by construction should lead to the VP solution.

The EVP sea ice dynamics model is thoroughly documented in [21], [20], [22] and [23] and the EAP dynamics in [60]. 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 [25] and [60].

The EVP numerical implementation in this code release is that of [22] and [23], with revisions to the numerical solver as in [6]. Details about the rEVP solver can be found in [35], [6], [28] and [30]. The implementation of the EAP sea ice dynamics into CICE is described in detail in [60].

The VP solver implementation mostly follows [33], with FGMRES [53] as the linear solver and GMRES as the preconditioner. Note that the VP solver has not yet been tested on the `tx1` grid.

The EVP, rEVP, EAP and VP approaches are all available with the B grid. However, at the moment, only the EVP and rEVP schemes are possible with the C grid.

Here we summarize the equations and direct the reader to the above references for details.

2.5.1 Momentum time stepping

EVP time discretization and solution

The momentum equation is discretized in time as follows, for the classic EVP approach. 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.

$$\begin{aligned}\underbrace{\left(\frac{m}{\Delta t_e} + \texttt{vrel}\cos\theta \ + C_b\right)}_{\texttt{cca}} u^{k+1} - \underbrace{\left(mf + \texttt{vrel}\sin\theta\right)}_{\texttt{ccb}} v^{l} = & \underbrace{\frac{\partial\sigma_{1j}^{k+1}}{\partial x_j}}_{\texttt{strintx}} + \underbrace{\tau_{ax} - mg\frac{\partial H_\circ}{\partial x}}_{\texttt{forcex}} \\ & + \texttt{vrel}\underbrace{\left(U_w\cos\theta - V_w\sin\theta\right)}_{\texttt{waterx}} + \frac{m}{\Delta t_e}u^k,\end{aligned} \tag{2.30}$$

$$\begin{aligned}\underbrace{\left(mf + \texttt{vrel}\sin\theta\right)}_{\texttt{ccb}} u^{l} + \underbrace{\left(\frac{m}{\Delta t_e} + \texttt{vrel}\cos\theta + C_b\right)}_{\texttt{cca}} v^{k+1} = & \underbrace{\frac{\partial\sigma_{2j}^{k+1}}{\partial x_j}}_{\texttt{strinty}} + \underbrace{\tau_{ay} - mg\frac{\partial H_\circ}{\partial y}}_{\texttt{forcey}} \\ & + \texttt{vrel}\underbrace{\left(U_w\sin\theta + V_w\cos\theta\right)}_{\texttt{watery}} + \frac{m}{\Delta t_e}v^k,\end{aligned} \tag{2.31}$$

where $\texttt{vrel} \cdot \texttt{waterx(y)} = \texttt{taux(y)}$ and the definitions of u^l and v^l vary depending on the grid.

As u and v are collocated on the B grid, u^l and v^l are respectively u^{k+1} and v^{k+1} such that this system of equations can be solved as follows. Define

$$\hat{u} = F_u + \tau_{ax} - mg\frac{\partial H_\circ}{\partial x} + \texttt{vrel}\left(U_w\cos\theta - V_w\sin\theta\right) + \frac{m}{\Delta t_e}u^k \tag{2.32}$$

$$\hat{v} = F_v + \tau_{ay} - mg\frac{\partial H_\circ}{\partial y} + \texttt{vrel}\left(U_w\sin\theta + V_w\cos\theta\right) + \frac{m}{\Delta t_e}v^k, \tag{2.33}$$

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

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

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

$$a = \frac{m}{\Delta t_e} + \texttt{vrel}\cos\theta + C_b \tag{2.34}$$

$$b = mf + \texttt{vrel}\sin\theta. \tag{2.35}$$

Note that the time discretization and solution method for the EAP is exactly the same as for the B grid EVP. More details on the EAP model are given in Section *Elastic-Anisotropic-Plastic*.

However, on the C grid, u and v are not collocated. When solving the u momentum equation for u^{k+1} (at the E point), $v^l = v^k_{int}$ where v^k_{int} is v^k from the surrounding N points interpolated to the E point. The same approach is used for the v momentum equation. With this explicit treatment of the off-diagonal terms [29], u^{k+1} and v^{k+1} are obtained by solving

$$u^{k+1} = \frac{\hat{u} + bv^k_{int}}{a}$$
$$v^{k+1} = \frac{\hat{v} - bu^k_{int}}{a}.$$

Revised EVP time discretization and solution

The revised EVP approach is based on a pseudo-time iterative scheme [35], [6], [28]. 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 (2.30) and (2.31) become

$$\begin{aligned}\frac{\beta^*(u^{k+1}-u^k)}{\Delta t_e} + \frac{m(u^{k+1}-u^n)}{\Delta t} + (\mathtt{vrel}\cos\theta + C_b)u^{k+1} - (mf + \mathtt{vrel}\sin\theta)v^l = \frac{\partial\sigma_{1j}^{k+1}}{\partial x_j} + \tau_{ax} \\ - mg\frac{\partial H_\circ}{\partial x} + \mathtt{vrel}(U_w\cos\theta - V_w\sin\theta),\end{aligned} \tag{2.36}$$

$$\begin{aligned}\frac{\beta^*(v^{k+1}-v^k)}{\Delta t_e} + \frac{m(v^{k+1}-v^n)}{\Delta t} + (\mathtt{vrel}\cos\theta + C_b)v^{k+1} + (mf + \mathtt{vrel}\sin\theta)u^l = \frac{\partial\sigma_{2j}^{k+1}}{\partial x_j} + \tau_{ay} \\ - mg\frac{\partial H_\circ}{\partial y} + \mathtt{vrel}(U_w\sin\theta + V_w\cos\theta),\end{aligned} \tag{2.37}$$

where β^* is a numerical parameter and u^n, v^n are the components of the previous time level solution. With $\beta = \beta^*\Delta t\,(m\Delta t_e)^{-1}$ [6], these equations can be written as

$$\begin{aligned}\underbrace{\left((\beta+1)\frac{m}{\Delta t} + \mathtt{vrel}\cos\theta\ + C_b\right)}_{\mathtt{cca}} u^{k+1} - \underbrace{(mf + \mathtt{vrel}\sin\theta)}_{\mathtt{ccb}} v^l = \underbrace{\frac{\partial\sigma_{1j}^{k+1}}{\partial x_j}}_{\mathtt{strintx}} + \underbrace{\tau_{ax} - mg\frac{\partial H_\circ}{\partial x}}_{\mathtt{forcex}} \\ + \mathtt{vrel}\underbrace{(U_w\cos\theta - V_w\sin\theta)}_{\mathtt{waterx}} + \frac{m}{\Delta t}(\beta u^k + u^n),\end{aligned} \tag{2.38}$$

$$\begin{aligned}\underbrace{(mf + \mathtt{vrel}\sin\theta)}_{\mathtt{ccb}} u^l + \underbrace{\left((\beta+1)\frac{m}{\Delta t} + \mathtt{vrel}\cos\theta + C_b\right)}_{\mathtt{cca}} v^{k+1} = \underbrace{\frac{\partial\sigma_{2j}^{k+1}}{\partial x_j}}_{\mathtt{strinty}} + \underbrace{\tau_{ay} - mg\frac{\partial H_\circ}{\partial y}}_{\mathtt{forcey}} \\ + \mathtt{vrel}\underbrace{(U_w\sin\theta + V_w\cos\theta)}_{\mathtt{watery}} + \frac{m}{\Delta t}(\beta v^k + v^n),\end{aligned} \tag{2.39}$$

At this point, the solutions u^{k+1} and v^{k+1} for the B or the C grids are obtained in the same manner as for the standard EVP approach (see Section *EVP time discretization and solution* for details).

Implicit (VP) time discretization and solution

In the VP approach, equation (2.29) is discretized implicitly using a Backward Euler approach, and stresses are not computed explicitly:

$$
\begin{aligned}
m\frac{(u^n - u^{n-1})}{\Delta t} &= \frac{\partial \sigma_{1j}^n}{\partial x_j} - \tau_{w,x}^n + \tau_{b,x}^n + mfv^n + r_x^n, \\
m\frac{(v^n - v^{n-1})}{\Delta t} &= \frac{\partial \sigma_{2j}^n}{\partial x_j} - \tau_{w,y}^n + \tau_{b,y}^n - mfu^n + r_y^n
\end{aligned}
\tag{2.40}
$$

where $r = (r_x, r_y)$ contains all terms that do not depend on the velocities u^n, v^n (namely the sea surface tilt and the wind stress). As the water drag, seabed stress and rheology term depend on the velocity field, the only unknowns in equation (2.40) are u^n and v^n.

Once discretized in space, equation (2.40) leads to a system of N nonlinear equations with N unknowns that can be concisely written as

$$\mathbf{A}(\mathbf{u})\mathbf{u} = \mathbf{b}(\mathbf{u}), \tag{2.41}$$

where $\mathbf{A}$ is an $N \times N$ matrix and $\mathbf{u}$ and $\mathbf{b}$ are vectors of size N. Note that we have dropped the time level index n. The vector $\mathbf{u}$ is formed by stacking first the u components, followed by the v components of the discretized ice velocity. The vector $\mathbf{b}$ is a function of the velocity vector $\mathbf{u}$ because of the water and seabed stress terms as well as parts of the rheology term that depend non-linearly on $\mathbf{u}$.

The nonlinear system (2.41) is solved using a Picard iteration method. Starting from a previous iterate $\mathbf{u}_{k-1}$, the nonlinear system is linearized by substituting $\mathbf{u}_{k-1}$ in the expression of the matrix $\mathbf{A}$ and the vector $\mathbf{b}$:

$$\mathbf{A}(\mathbf{u}_{k-1})\mathbf{u}_k = \mathbf{b}(\mathbf{u}_{k-1}) \tag{2.42}$$

The resulting linear system is solved using the Flexible Generalized Minimum RESidual (FGMRES, [53]) method and this process is repeated iteratively.

The maximum number of Picard iterations can be set using the namelist flag `maxits_nonlin`. The relative tolerance for the Picard solver can be set using the namelist flag `reltol_nonlin`. The Picard iterative process stops when $\left\|\mathbf{u}_k\right\|_2 < \texttt{reltol\_nonlin} \cdot \left\|\mathbf{u}_0\right\|_2$ or when `maxits_nonlin` is reached.

Parameters for the FGMRES linear solver and the preconditioner can be controlled using additional namelist flags (see *dynamics_nml*).

2.5.2 Surface stress terms

The formulation for the wind stress is described in Icepack Documentation. Below, some details about the ice-ocean stress and the seabed stress are given.

Ice-Ocean stress

At the end of 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 [17] is included in **ice_dyn_shared.F90** but is currently commented out, pending further testing.

Seabed stress

CICE includes two options for calculating the seabed stress, i.e. the term in the momentum equation that represents the interaction between grounded ice keels and the seabed. The seabed stress can be activated by setting `seabed_stress` to true in the namelist. The seabed stress (or basal stress) parameterization of [34] is chosen if `seabed_stress_method` = `LKD` while the approach based on the probability of contact between the ice and the seabed is used if `seabed_stress_method` = `probabilistic`.

For both parameterizations, the components of the seabed stress are expressed as $\tau_{bx} = C_b u$ and $\tau_{by} = C_b v$, where C_b is a seabed stress coefficient.

The two parameterizations differ in their calculation of the C_b coefficients.

Note that the user must provide a bathymetry field for using these grounding schemes. It is suggested to have a bathymetry field with water depths larger than 5 m that represents well shallow water (less than 30 m) regions such as the Laptev Sea and the East Siberian Sea.

Seabed stress based on linear keel draft (LKD)

This parameterization for the seabed stress is described in [34]. It assumes that the largest keel draft varies linearly with the mean thickness in a grid cell (i.e. sea ice volume). The C_b coefficients are expressed as

$$C_b = k_2 \max[0, (h - h_c)] e^{-\alpha_b * (1-a)} (\sqrt{u^2 + v^2} + u_0)^{-1}, \tag{2.43}$$

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 α_b is a parameter to ensure that the seabed stress quickly drops when the ice concentration is smaller than 1. In the code, $k_2 \max[0, (h - h_c)] e^{-\alpha_b * (1-a)}$ is defined as T_b.

On the B grid, the quantities h, a and h_c are calculated at the U point and are referred to as h_u, a_u and h_{cu}. They are respectively given by

$$h_u = \max[v_i(i,j), v_i(i+1,j), v_i(i,j+1), v_i(i+1,j+1)], \tag{2.44}$$

$$a_u = \max[a_i(i,j), a_i(i+1,j), a_i(i,j+1), a_i(i+1,j+1)], \tag{2.45}$$

$$h_{cu} = a_u h_{wu} / k_1, \tag{2.46}$$

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 (2.43), the seabed stress components are non-zero only when $h_u > h_{cu}$.

As u and v are not collocated on the C grid, T_b is calculated at E and N points. For example, at the E point, h_e, a_e and h_{ce} are respectively

$$h_e = \max[v_i(i,j), v_i(i+1,j)], \tag{2.47}$$

$$a_e = \max[a_i(i,j), a_i(i+1,j)], \tag{2.48}$$

$$h_{ce} = a_e h_{we} / k_1, \tag{2.49}$$

where $h_{we} = \min[h_w(i,j), h_w(i+1,j)]$. Similar calculations are done at the N points.

To prevent unrealistic grounding, T_b is set to zero when h_{wu} is larger than 30 m (same idea on the C grid depending on h_{we} and h_{wn}). This maximum value is chosen based on observations of large keels in the Arctic Ocean [1].

The maximum seabed stress depends on the weight 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`.

Seabed stress based on probabilistic approach

This more sophisticated grounding parameterization computes the seabed stress based on the probability of contact between the ice thickness distribution (ITD) and the seabed [11]. Multi-thickness category models such as CICE typically use a few thickness categories (5-10). This crude representation of the ITD does not resolve the tail of the ITD, which is crucial for grounding events.

To represent the tail of the distribution, the simulated ITD is converted to a positively skewed probability function $f(x)$ with x the sea ice thickness. The mean and variance are set equal to the ones of the original ITD. A log-normal distribution is used for $f(x)$.

It is assumed that the bathymetry y (at the 't' point) follows a normal distribution $b(y)$. The mean of $b(y)$ comes from the user's bathymetry field and the standard deviation σ_b is currently fixed to 2.5 m. Two possible improvements would be to specify a distribution based on high resolution bathymetry data and to take into account variations of the water depth due to changes in the sea surface height.

Assuming hydrostatic balance and neglecting the impact of snow, the draft of floating ice of thickness x is $D(x) = \rho_i x/\rho_w$ where ρ_i is the sea ice density. Hence, the probability of contact (P_c) between the ITD and the seabed is given by

$$P_c = \int_0^{\inf} \int_0^{D(x)} g(x)b(y)dydx.$$

T_b is first calculated at the T point (referred to as T_{bt}). T_{bt} depends on the weight of the ridge in excess of hydrostatic balance. The parameterization first calculates

$$T_{bt}^* = \mu_s g \int_0^{\inf} \int_0^{D(x)} (\rho_i x - \rho_w y) g(x)b(y)dydx, \tag{2.50}$$

and then obtains T_{bt} by multiplying T_{bt}^* by $e^{-\alpha_b * (1-a_i)}$ (similar to what is done for `seabed_stress_method = LKD`).

To calculate T_{bt}^* in equation (2.50), $f(x)$ and $b(y)$ are discretized using many small categories (100). $f(x)$ is discretized between 0 and 50 m while $b(y)$ is truncated at plus and minus three σ_b. $f(x)$ is also modified by setting it to zero after a certain percentile of the log-normal distribution. This percentile, which is currently set to 99.7%, notably affects the simulation of landfast ice and is used as a tuning parameter. Its impact is similar to the one of the parameter k_1 for the LKD method.

On the B grid, T_b at the U point is calculated from the T point values around it according to

$$T_{bu} = \max[T_{bt}(i,j), T_{bt}(i+1,j), T_{bt}(i,j+1), T_{bt}(i+1,j+1)]. \tag{2.51}$$

Following again the LKD method, the seabed stress coefficients are finally expressed as

$$C_b = T_{bu}(\sqrt{u^2+v^2}+u_0)^{-1}. \tag{2.52}$$

On the C grid, T_b is needs to be calculated at the E and N points. T_{be} and T_{bn} are respectively given by

$$T_{be} = \max[T_{bt}(i,j), T_{bt}(i+1,j)], \tag{2.53}$$

$$T_{bn} = \max[T_{bt}(i,j), T_{bt}(i,j+1)]. \tag{2.54}$$

The C_b are different at the E and N points and are respectively $T_{be}(\sqrt{u^2+v_{int}^2}+u_0)^{-1}$ and $T_{bn}(\sqrt{u_{int}^2+v^2}+u_0)^{-1}$ where v_{int} (u_{int}) is v (u) interpolated to the E (N) point.

2.5.3 Rheology

For convenience we formulate the stress tensor σ in terms of $\sigma_1 = \sigma_{11} + \sigma_{22}$ (`stressp`), $\sigma_2 = \sigma_{11} - \sigma_{22}$ (`stressm`), and introduce the divergence, D_D, and the horizontal tension and shearing strain rates, D_T and D_S respectively:

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

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

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

where

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

Note that σ_1 and σ_2 are not to be confused with the normalized principal stresses, $\sigma_{n,1}$ and $\sigma_{n,2}$ (`sig1` and `sig2`), which are defined as:

$$\sigma_{n,1}, \sigma_{n,2} = \frac{1}{P}\left(\frac{\sigma_1}{2} \pm \sqrt{\left(\frac{\sigma_2}{2}\right)^2 + \sigma_{12}^2}\right)$$

where P is the ice strength.

In addition to the normalized principal stresses, CICE can output 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 (σ_{11}, σ_{22}) multiplied by -1 and is therefore simply equal to $-\sigma_1/2$.

Viscous-Plastic

The VP constitutive law is given by

$$\sigma_{ij} = 2\eta\dot{\epsilon}_{ij} + (\zeta - \eta)D_D - P_R\frac{\delta_{ij}}{2} \tag{2.55}$$

where η and ζ are the bulk and shear viscosities and P_R is a "replacement pressure" (see [13], for example), which serves to prevent residual ice motion due to spatial variations of the ice strength P when the strain rates are exactly zero.

An elliptical yield curve is used, with the viscosities given by

$$\zeta = \frac{P(1 + k_t)}{2\Delta}, \tag{2.56}$$

$$\eta = e_g^{-2}\zeta, \tag{2.57}$$

where

$$\Delta = \left[D_D^2 + \frac{e_f^2}{e_g^4}\left(D_T^2 + D_S^2\right)\right]^{1/2}. \tag{2.58}$$

When the deformation Δ tends toward zero, the viscosities tend toward infinity. To avoid this issue, Δ needs to be limited and is replaced by Δ^* in equation (2.56). Two methods for limiting Δ (or for capping the viscosities) are available in the code. If the namelist parameter `capping_method` is set to `max`, $\Delta^* = max(\Delta, \Delta_{min})$ [15] while with `capping_method` set to `sum`, the smoother formulation $\Delta^* = (\Delta + \Delta_{min})$ of [31] is used.

The ice strength P is a function of the ice thickness distribution as described in the Icepack Documentation.

Two other modifications to the standard VP rheology of [15] are available. First, following the approach of [3] (see also [34]), the elliptical yield curve can be modified such that the ice has isotropic tensile strength. The tensile strength is expressed as a fraction of P, that is $k_t P$ where k_t should be set to a value between 0 and 1 (this can be changed at runtime with the namelist parameter `Ktens`).

Second, while e_f is the ratio of the major and minor axes of the elliptical yield curve, the parameter e_g characterizes the plastic potential, i.e. another ellipse that decouples the flow rule from the yield curve ([46]). e_f and e_g are respectively called `e_yieldcurve` and `e_plasticpot` in the code and can be set in the namelist. The plastic potential can lead to more realistic fracture angles between linear kinematic features. [46] suggest to set e_f to a value larger than 1 and to have $e_g < e_f$.

By default, the namelist parameters are set to $e_f = e_g = 2$ and $k_t = 0$ which correspond to the standard VP rheology.

There are four options in the code for solving the sea ice momentum equation with a VP formulation: the standard EVP approach, a 1d EVP solver, the revised EVP approach and an implicit Picard solver. The choice of the capping method for the viscosities and the modifications to the yield curve and to the flow rule described above are available for these four different solution methods. Note that only the EVP and revised EVP methods are currently available if one chooses the C grid.

Elastic-Viscous-Plastic

In the EVP model the internal stress tensor is determined from a regularized version of the VP constitutive law (2.55). The constitutive law is therefore

$$\frac{1}{E}\frac{\partial\sigma_1}{\partial t} + \frac{\sigma_1}{2\zeta} + \frac{P_R}{2\zeta} = D_D, \tag{2.59}$$

$$\frac{1}{E}\frac{\partial\sigma_2}{\partial t} + \frac{\sigma_2}{2\eta} = D_T, \tag{2.60}$$

$$\frac{1}{E}\frac{\partial\sigma_{12}}{\partial t} + \frac{\sigma_{12}}{2\eta} = \frac{1}{2}D_S, \tag{2.61}$$

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 $E_\circ$ (elasticDamp) is a tunable parameter less than one. Including the modification proposed by [6] for equations (2.60) and (2.61) in order to improve numerical convergence, the stress equations become

$$\begin{aligned}
\frac{\partial\sigma_1}{\partial t} + \frac{\sigma_1}{2T} + \frac{P_R}{2T} &= \frac{\zeta}{T}D_D, \\
\frac{\partial\sigma_2}{\partial t} + \frac{\sigma_2}{2T} &= \frac{\eta}{T}D_T, \\
\frac{\partial\sigma_{12}}{\partial t} + \frac{\sigma_{12}}{2T} &= \frac{\eta}{2T}D_S.
\end{aligned}$$

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

$$\begin{aligned}
\frac{(\sigma_1^{k+1} - \sigma_1^k)}{\Delta t_e} + \frac{\sigma_1^{k+1}}{2T} + \frac{P_R^k}{2T} &= \frac{\zeta^k}{T}D_D^k, \\
\frac{(\sigma_2^{k+1} - \sigma_2^k)}{\Delta t_e} + \frac{\sigma_2^{k+1}}{2T} &= \frac{\eta^k}{T}D_T^k, \\
\frac{(\sigma_{12}^{k+1} - \sigma_{12}^k)}{\Delta t_e} + \frac{\sigma_{12}^{k+1}}{2T} &= \frac{\eta^k}{2T}D_S^k,
\end{aligned} \tag{2.62}$$

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. Choices of the parameters used to define E, T and Δt_e are discussed in Sections *Revised EVP approach* and *Choosing an appropriate time step*.

On the B grid, the stresses σ_1, σ_2 and σ_{12} are collocated at the U point. To calculate these stresses, the viscosities ζ and η and the replacement pressure P_R are also defined at the U point.

However, on the C grid, σ_1 and σ_2 are collocated at the T point while σ_{12} is defined at the U point. During a subcycling step, ζ, η and P_R are first calculated at the T point. To do so, Δ given by equation (2.58) is calculated following the approach of [6] (see also [29] for details). With this approach, D_S^2 at the T point is obtained by calculating D_S^2 at the U points and interpolating these values to the T point. As σ_{12} is calculated at the U point, η also needs to be computed as these locations. If `visc_method` in the namelist is set to `avg_zeta` (the default value), η at the U point is obtained by interpolating T point values to this location. This corresponds to the approach used by [6] and the one associated with the C1 configuration of [29]. On the other hand, if `visc_method = avg_strength`, the strength P calculated at T points is interpolated to the U point and Δ is calculated at the U point in order to obtain η following equations (2.56) and (2.57). This latter approach is the one used in the C2 configuration of [29].

1d EVP solver

The standard EVP solver iterates hundreds of times, where each iteration includes a communication through MPI and a limited number of calculations. This limits how much the solver can be optimized as the speed is primarily determined by the communication. The 1d EVP solver avoids the communication by utilizing shared memory, which removes the requirement for calls to the MPI communicator. As a consequence of this the potential scalability of the code is improved. The performance is best on shared memory but the solver is also functional on MPI and hybrid MPI/OpenMP setups as it will run on the master processor alone.

The scalability of geophysical models is in general terms limited by the memory usage. In order to optimize this the 1d EVP solver solves the same equations that are outlined in the section *Elastic-Viscous-Plastic* but it transforms all matrices to vectors (1d matrices) as this compiles better with the computer hardware. The vectorization and the contiguous placement of arrays in the memory makes it easier for the compiler to optimize the code and pass pointers instead of copying the vectors. The 1d solver is not supported for tripole grids and the code will abort if this combination is attempted.

Revised EVP approach

Introducing the numerical parameter $\alpha = 2T\Delta t_e^{-1}$ [6], the stress equations in (2.62) become

$$\begin{aligned} \alpha(\sigma_1^{k+1} - \sigma_1^k) + \sigma_1^k + P_R^k &= 2\zeta^k D_D^k, \\ \alpha(\sigma_2^{k+1} - \sigma_2^k) + \sigma_2^k &= 2\eta^k D_T^k, \\ \alpha(\sigma_{12}^{k+1} - \sigma_{12}^k) + \sigma_{12}^k &= \eta^k D_S^k, \end{aligned}$$

where as opposed to the classic EVP, the second term in each equation is at iteration k [6]. Also, contrary to the classic EVP, Δt_e times the number of subcycles (or iterations) does not need to be equal to the advective time step Δt. 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 α is `arlx` and β is `brlx` (introduced in Section *Revised EVP time discretization and solution*). 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 $\alpha = \beta$ [28].

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 π, and the positive coefficients ω_1 and ω_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 $\mathbf{A}$,

$$\frac{D\mathbf{A}}{Dt} = \mathbf{F}_{iso}(\mathbf{A}) + \mathbf{F}_{frac}(\mathbf{A}, \boldsymbol{\sigma}), \tag{2.63}$$

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 [65], based on laboratory experiments by [54] 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 σ_1 and σ_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 [65]). ϕ 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 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 (2.63) 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 (2.63) 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 (2.63) that are given in [65] and [60]. 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 ϕ, we make no assumption about the size of the diamonds so that formally the theory is scale invariant.

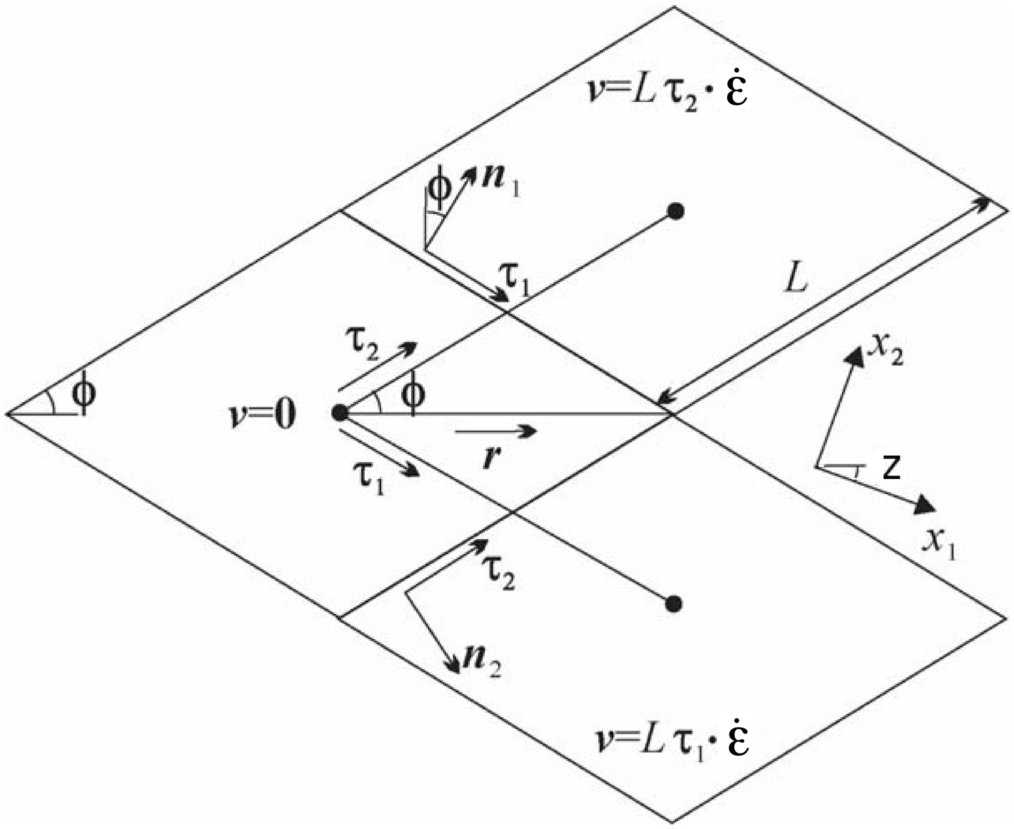


Fig. 3: Diamond-shaped floes

As described in greater detail in [65], 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

$$\boldsymbol{\sigma}^b(\mathbf{r},h) = P_r(h)\boldsymbol{\sigma}_r^b(\mathbf{r}) + P_s(h)\boldsymbol{\sigma}_s^b(\mathbf{r}), \tag{2.64}$$

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

$$\boldsymbol{\sigma}^{EAP}(h) = P_r(h)\int_{\mathbb{S}} \vartheta(\mathbf{r})\left[\boldsymbol{\sigma}_r^b(\mathbf{r}) + k\boldsymbol{\sigma}_s^b(\mathbf{r})\right]d\mathbf{r} \tag{2.65}$$

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,

$$\frac{\partial\sigma_1}{\partial t} + \frac{\sigma_1}{2T} = \frac{\sigma_1^{EAP}}{2T}, \tag{2.66}$$

$$\frac{\partial\sigma_2}{\partial t} + \frac{\sigma_2}{2T} = \frac{\sigma_2^{EAP}}{2T}, \tag{2.67}$$

$$\frac{\partial\sigma_{12}}{\partial t} + \frac{\sigma_{12}}{2T} = \frac{\sigma_{12}^{EAP}}{2T}, \tag{2.68}$$

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 (2.66)–(2.68) for discrete values of the largest eigenvalue of the structure tensor, $\frac{1}{2} \leq A_1 \leq 1$, the angle $0 \leq \theta \leq 2\pi$, and the angle $-\pi/2 \leq y \leq \pi/2$ between the major principal axis of the strain rate tensor and the structure tensor [60]. 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, Δt_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 [64],

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

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

CHAPTER

THREE

USER GUIDE

3.1 Implementation

CICE is written in FORTRAN90 and runs on platforms using UNIX, LINUX, and other operating systems. The current coding standard is Fortran2003 with use of Fortran2008 feature CONTIGUOUS in the 1d evp solver. 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/cicedyn/
routines associated with the dynamics core

cicecore/drivers/
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

dot files
various files that begin with . and store information about the git repository or other tools.

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 of the original implementation is specialized for a generalized orthogonal B-grid as in [42] or [55]. Figure *Schematic of CICE B-grid.* is a schematic of CICE B-grid. This cell with the tracer point $t(i,j)$ in the middle is referred to as T-cell. The ice and snow area, volume and energy are given at the t-point. The velocity $\mathbf{u}(i,j)$ associated with $t(i,j)$ is defined in the northeast (NE) corner. The other corners of the T-cell are northwest (NW), southwest (SW) and southeast (SE). The lengths of the four edges of the T-cell are respectively HTN, HTW, HTS and HTE for the northern, western, southern and eastern edges. The lengths of the T-cell through the middle are respectively dxT and dyT along the x and y axis.

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 internal ice stress tensor takes four different values within a grid cell with the B-grid implementation; bilinear approximations are used for the stress tensor and the ice velocity across the cell, as described in [22]. This tends to avoid the grid decoupling problems associated with the B-grid.

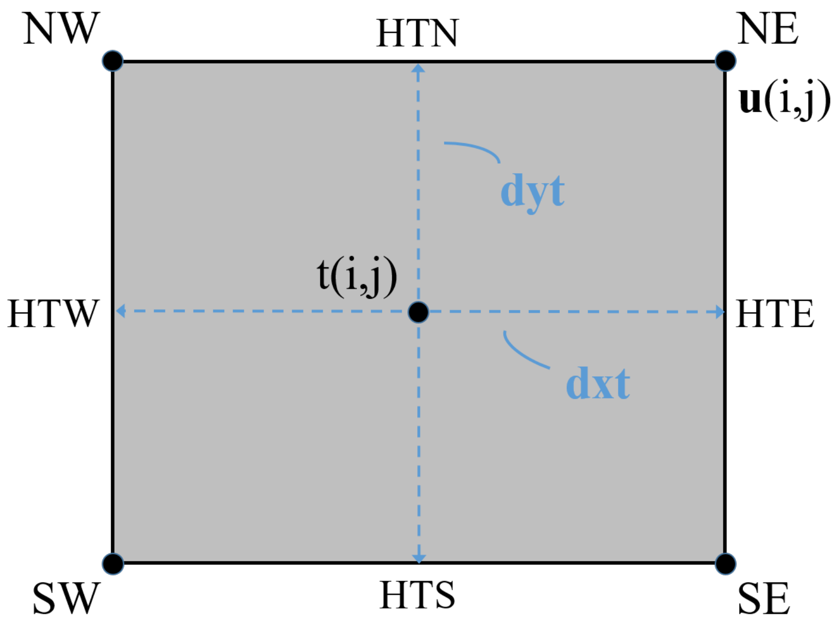


Fig. 1: Schematic of CICE B-grid.

The ability to solve on the C and CD grids was added later. With the C-grid, the u velocity points are located on the E edges and the v velocity points are located on the N edges of the T cell rather than at the T cell corners. On the CD-grid, the u and v velocity points are located on both the N and E edges. To support this capability, N and E grids were added to the existing T and U grids, and the N and E grids are defined at the northern and eastern edge of the T cell. This is shown in Figure *Schematic of CICE CD-grid.*.

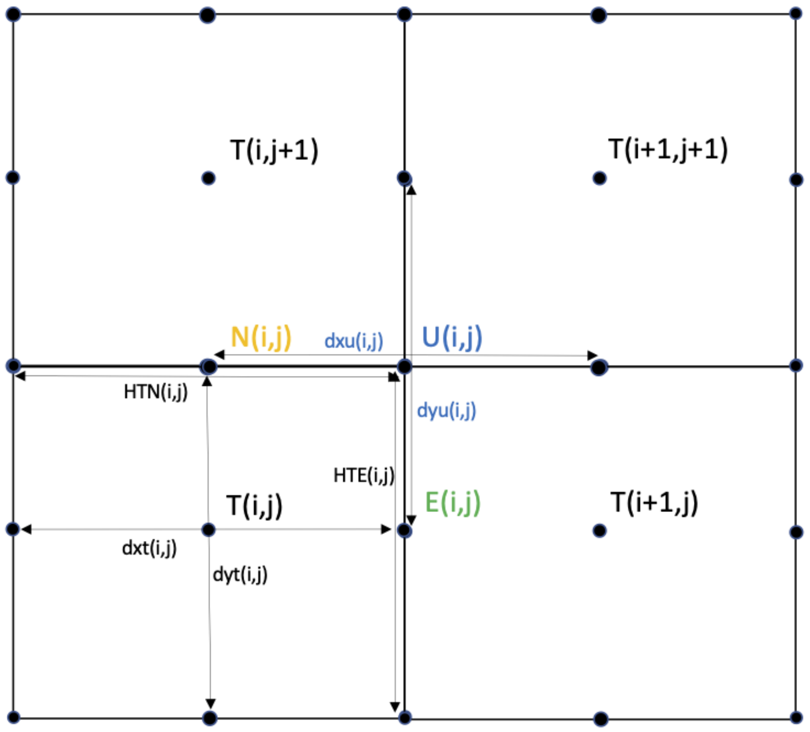


Fig. 2: Schematic of CICE CD-grid.

The user has several ways to initialize the grid: *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 ⟨3°⟩ POP grid and land mask; **global_gx1.grid** and **global_gx1.kmt** contain the ⟨1°⟩ grid and land mask, and **global_tx1.grid** and **global_tx1.kmt** contain the ⟨1°⟩ POP tripole grid and land mask. These are binary unformatted, direct access, Big Endian files.

The input grid file for the B-grid and CD-grid is identical. That file contains each cells' HTN, HTE, ULON, ULAT, and kmt value. From those variables, the longitude, latitude, grid lengths (dx and dy), areas, and masks can be derived for all grids. Table *Primary CICE Prognostic Grid Variable Names* lists the primary prognostic grid variable names on the different grids.

Table 1: Primary CICE Prognostic Grid Variable Names

variable	T	U	N	E
longitude	TLON	ULON	NLON	ELON
latitude	TLAT	ULAT	NLAT	ELAT
dx	dxT	dxU	dxN	dxE
dy	dyT	dyU	dyN	dyE
area	tarea	uarea	narea	earea
mask (logical)	tmask	umask	nmask	emask
mask (real)	hm	uvm	npm	epm

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` $\times$`ny_global`, while the subdomains used in the block distribution are `nx_block` $\times$`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. `nghost` is a hardcoded parameter in **ice_blocks.F90**. While the halo code has been implemented to support arbitrary sized halos, `nghost` is set to 1 and has not been formally tested on larger halos.

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` $\times$`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. The model is parallelized over blocks for both MPI and OpenMP. Some suggested combinations for 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 tentative `max_blocks` on the fly.

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 the namelist variable `debug_blocks` 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. This `debug_blocks` variable should be used carefully as there may be hundreds or thousands of blocks to print and this information should be needed only rarely. `debug_blocks` can be set to true using the `debugblocks` option with **cice.setup**. 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`.

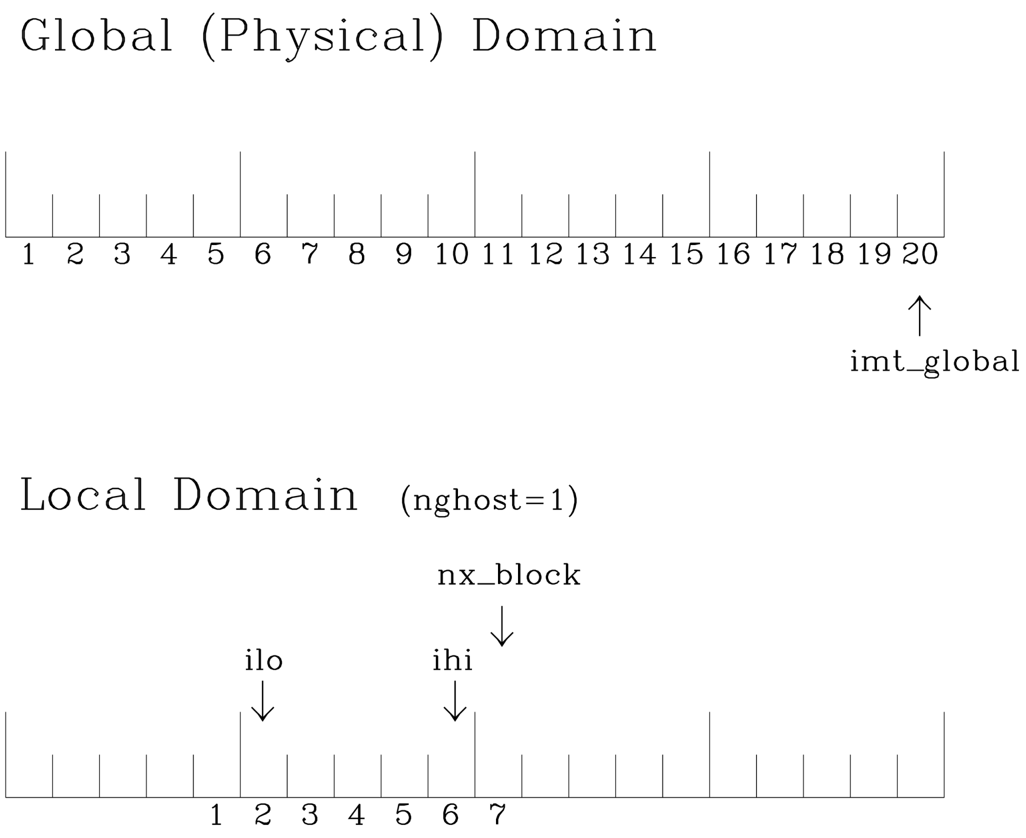


Fig. 3: Grid parameters

The namelist `add_mpi_barriers` can be set to `.true.` to help throttle communication for communication intensive configurations. This may slow the code down a bit. These barriers have been added to a few select locations, but it's possible others may be needed. As a general rule, `add_mpi_barriers` should be `.false.`.

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 schematics below, symbols A-H represent grid points from 1:nx_global at a given j index and the setup of the tripole seam is depicted within a few rows of the seam.

Table 2: Tripole (u-fold) Grid Schematic

global j index	grid point IDs (i index)								global j index source
ny_global+2	H	G	F	E	D	C	B	A	ny_global-1
ny_global+1	H	G	F	E	D	C	B	A	ny_global
ny_global	A	B	C	D	E	F	G	H	
ny_global-1	A	B	C	D	E	F	G	H	

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 $n - 2$.

Table 3: TripoleT (t-fold) Grid Schematic

global j index	grid point IDs (i index)									global j index source
ny_global+2		H	G	F	E	D	C	B	A	ny_global-2
ny_global+1		H	G	F	E	D	C	B	A	ny_global-1
ny_global	A	BH	CG	DF	E	FD	GC	HB		
ny_global-1	A	B	C	D	E	F	G	H		
ny_global-2	A	B	C	D	E	F	G	H		

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. Halos of center, east, north, and northeast points are supported, and each requires slightly different halo indexing across the tripole seam.

Rectangular grids

Rectangular test grids can be defined for CICE. They are generated internally and defined by several namelist settings including `grid_type = rectangular`, `nx_global`, `ny_global`, `dx_rect`, `dy_rect`, `lonrefrect`, and `latrefrect`. Forcing and initial condition can be set via namelists `atm_data_type`, `ocn_data_type`, `ice_data_type`, `ice_data_conc`, `ice_data_dist`. Variable grid spacing is also supported with the namelist settings `scale_dxdy` which turns on the option, and `dxscale` and `dyscale` which sets the variable grid scaling factor. Values of 1.0 will produced constant grid spacing. For rectangular grids, `lonrefrect` and `latrefrect` define the lower left longitude and latitude value of the grid, `dx_rect` and `dy_rect` define the base grid spacing, and `dxscale` and `dyscale` provide the grid space scaling. The base spacing is set in the center of the rectangular domain and the scaling is applied symetrically outward as a multiplicative factor in the x and y directions.

Several predefined rectangular grids are available in CICE with **cice.setup –grid** including `gbox12`, `gbox80`, `gbox128`, and `gbox180` where 12, 80, 128, and 180 are the number of gridcells in each direction. Several predefined options also exist, set with **cice.setup –set**, to establish varied idealized configurations of box tests including `box2001`, `boxadv`, `boxchan`, `boxchan1e`, `boxchan1n`, `boxnodyn`, `boxrestore`, `boxslotcyl`, and `boxopen`, `boxclosed`, and `boxforcee`. See **cice.setup –help** for a current list of supported settings.

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.

Boundary conditions are defined by the `ns_boundary_type` and `ew_boundary_type` namelist inputs. Valid values are `open` and `cyclic`. In addition, `tripole` and `tripoleT` are options for the `ns_boundary_type`. Closed boundary conditions are not supported currently. The domain can be physically closed with the `close_boundaries` namelist which forces a land mask on the boundary with a two gridcell depth. Where the boundary is land, the boundary_type settings play no role. For example, in the displaced-pole grids, at least one row of grid cells along the north and south boundaries is land. Along the east/west domain boundaries not masked by land, periodic conditions wrap the domain around the globe. In this example, the appropriate namelist settings are `nsboundary_type` = `open`, `ew_boundary_type` = `cyclic`, and `close_boundaries` = `.false.`.

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 (on the T-grid), with 0 representing land and 1 representing ocean cells. Corresponding masks for the U, N, and E grids are given by

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

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

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

The logical masks `tmask`, `umask`, `nmask`, and `emask` (which correspond to the real masks `hm`, `uvm`, `npm`, and `epm` 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 `iceTmask` and `iceUmask` 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 `iceTmask` and `iceUmask` are for T-cells and U-cells, respectively. Note that the ice extent masks `iceEmask` and `iceNmask` are also defined when using the C or CD grid.

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.

Interpolating between grids

Fields in CICE are generally defined at particular grid locations, such as T cell centers, U corners, or N or E edges. These are assigned internally in CICE based on the `grid_ice` namelist variable. Forcing/coupling fields are also associated with a specific set of grid locations that may or may not be the same as on the internal CICE model grid. The namelist variables `grid_atm` and `grid_ocn` define the forcing/coupling grids. The `grid_ice`, `grid_atm`, and `grid_ocn` variables are independent and take values like `A`, `B`, `C`, or `CD` consistent with the Arakawa grid convention [2]. The relationship between the grid system and the internal grids is shown in *Grid System and Type Definitions*.

Table 4: Grid System and Type Definitions

grid system	thermo grid	u dynamic grid	v dynamic grid
A	T	T	T
B	T	U	U
C	T	E	N
CD	T	N+E	N+E

For all grid systems, thermodynamic variables are always defined on the T grid for the model and model forcing/coupling fields. However, the dynamics u and v fields vary. In the CD grid, there are twice as many u and v fields as on the other grids. Within the CICE model, the variables grid_ice_thrm, grid_ice_dynu, grid_ice_dynv, grid_atm_thrm, grid_atm_dynu, grid_atm_dynv, grid_ocn_thrm, grid_ocn_dynu, and grid_ocn_dynv are character strings (T, U, N, E , NE) derived from the grid_ice, grid_atm, and grid_ocn namelist values.

The CICE model has several internal methods that will interpolate (a.k.a. map or average) fields on (T, U, N, E, NE) grids to (T, U, N, E). An interpolation to an identical grid results in a field copy. The generic interface to this method is grid_average_X2Y, and there are several forms.

```
subroutine grid_average_X2Y(type,work1,grid1,work2,grid2)
  character(len=*)     , intent(in)  :: type           ! mapping type (S, A, F)
  real (kind=dbl_kind), intent(in)  :: work1(:,:,:)   ! input field(nx_block, ny_block,␣
↪max_blocks)
  character(len=*)     , intent(in)  :: grid1          ! work1 grid (T, U, N, E)
  real (kind=dbl_kind), intent(out) :: work2(:,:,:)   ! output field(nx_block, ny_block,␣
↪max_blocks)
  character(len=*)     , intent(in)  :: grid2          ! work2 grid (T, U, N, E)
```

where type is an interpolation type with the following valid values,

type = S is a normalized, masked, area-weighted interpolation

$$work2 = \frac{\sum_{i=1}^{n}(M_{1i}A_{1i}work1_i)}{\sum_{i=1}^{n}(M_{1i}A_{1i})}$$

type = A is a normalized, unmasked, area-weighted interpolation

$$work2 = \frac{\sum_{i=1}^{n}(A_{1i}work1_i)}{\sum_{i=1}^{n}(A_{1i})}$$

type = F is a normalized, unmasked, conservative flux interpolation

$$work2 = \frac{\sum_{i=1}^{n}(A_{1i}work1_i)}{n * A_2}$$

with A defined as the appropriate gridcell area and M as the gridcell mask. Another form of the grid_average_X2Y is

```
subroutine grid_average_X2Y(type,work1,grid1,wght1,mask1,work2,grid2)
  character(len=*)     , intent(in)  :: type           ! mapping type (S, A, F)
  real (kind=dbl_kind), intent(in)  :: work1(:,:,:)   ! input field(nx_block, ny_block,␣
↪max_blocks)
  real (kind=dbl_kind), intent(in)  :: wght1(:,:,:)   ! input weight(nx_block, ny_block,␣
↪max_blocks)
  real (kind=dbl_kind), intent(in)  :: mask1(:,:,:)   ! input mask(nx_block, ny_block,␣
↪max_blocks)
  character(len=*)     , intent(in)  :: grid1          ! work1 grid (T, U, N, E)
  real (kind=dbl_kind), intent(out) :: work2(:,:,:)   ! output field(nx_block, ny_block,␣
↪max_blocks)
  character(len=*)     , intent(in)  :: grid2          ! work2 grid (T, U, N, E)
```

In this case, the input arrays *wght1* and *mask1* are used in the interpolation equations instead of gridcell area and mask. This version allows the user to define the weights and mask explicitly. This implementation is supported only for type = S or A interpolations.

A final form of the grid_average_X2Y interface is

```
subroutine grid_average_X2Y(type,work1a,grid1a,work1b,grid1b,work2,grid2)
  character(len=*)    , intent(in)  :: type            ! mapping type (S, A, F)
  real (kind=dbl_kind), intent(in)  :: work1a(:,:,:)   ! input field(nx_block, ny_block,␣
↪max_blocks)
  character(len=*)    , intent(in)  :: grid1a          ! work1 grid (N, E)
  real (kind=dbl_kind), intent(in)  :: work1b(:,:,:)   ! input field(nx_block, ny_block,␣
↪max_blocks)
  character(len=*)    , intent(in)  :: grid1b          ! work1 grid (N, E)
  real (kind=dbl_kind), intent(out) :: work2(:,:,:)    ! output field(nx_block, ny_block,␣
↪max_blocks)
  character(len=*)    , intent(in)  :: grid2           ! work2 grid (T, U)
```

This version supports mapping from an NE grid to a T or U grid. In this case, the 1a arguments are for either the *N* or *E* field and the 1b arguments are for the complementary field (E or N respectively). At present, only S type mappings are supported with this interface.

In all cases, the work1, wght1, and mask1 input arrays should have correct halo values when called. Examples of usage can be found in the source code, but the following example maps the uocn and vocn fields from their native forcing/coupling grid to the U grid using a masked, area-weighted, average method.

```
call grid_average_X2Y('S', uocn, grid_ocn_dynu, uocnU, 'U')
call grid_average_X2Y('S', vocn, grid_ocn_dynv, vocnU, 'U')
```

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. The code is parallelized over blocks for both MPI and OpenMP. 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.

Like MPI, load balance of blocks across threads is important for efficient performance. Most of the OpenMP threading is implemented with SCHEDULE(runtime), so the OMP_SCHEDULE env variable can be used to set the OpenMPI schedule. The default OMP_SCHEDULE setting is defined by the variable ICE_OMPSCHE in **cice.settings**. OMP_SCHEDULE values of "STATIC,1" and "DYNAMIC,1" are worth testing. The OpenMP implementation in CICE is constantly under review, but users should validate results and performance on their machine. CICE should be bit-for-bit with different block sizes, different decompositions, different MPI task counts, and different OpenMP threads. Finally, we recommend the OMP_STACKSIZE env variable should be set to 32M or greater.

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.

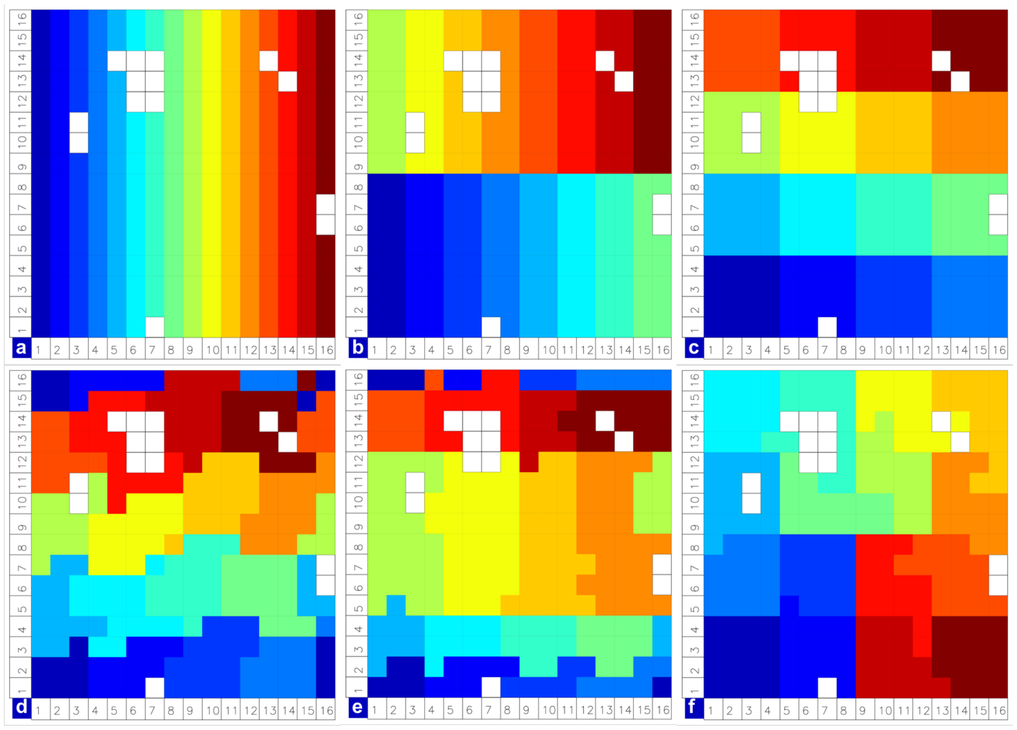


Fig. 4: Distribution options

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.

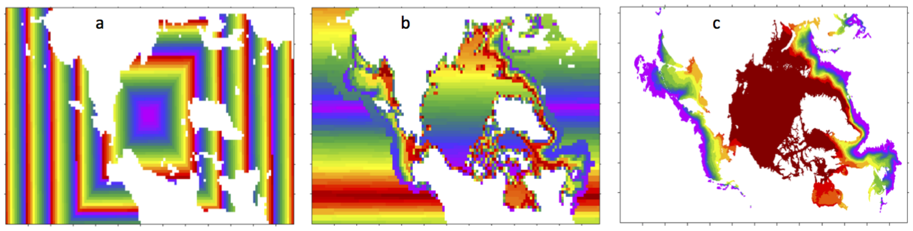


Fig. 5: Decomposition options

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).

`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. Blocks that contain 100% land grid cells are eliminated with 'block'. The 'blockall' option is identical to 'block' but does not do land block elimination. 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.

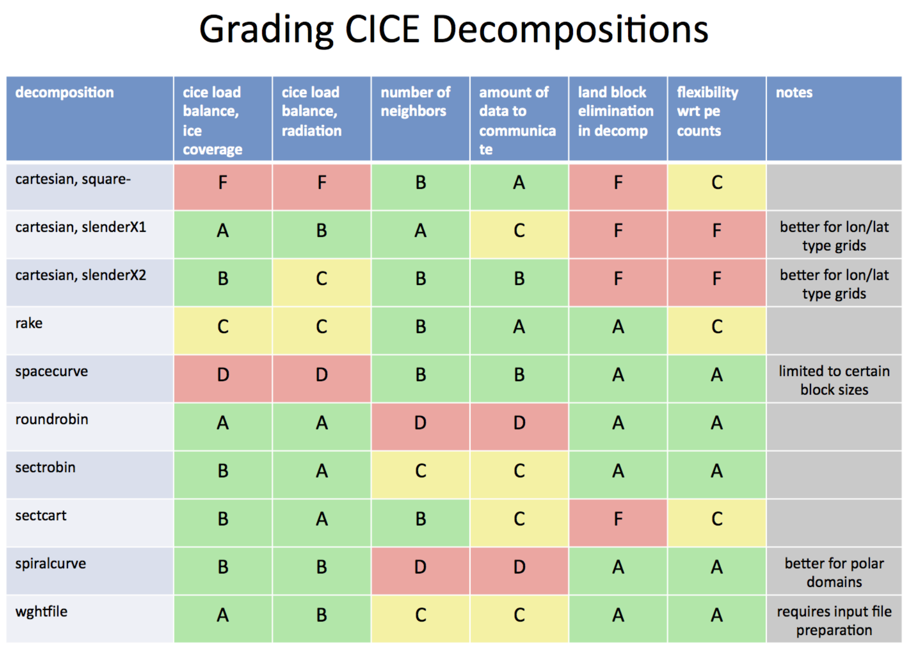
Figure *Scorecard* shows the scorecard for block distribution choices in CICE, courtesy T. Craig. For more information, see [9] or http://www.cesm.ucar.edu/events/workshops/ws.2012/presentations/sewg/craig.pdf

The `maskhalo` options in the namelist improve performance by removing unnecessary halo communications where 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 20×192 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.

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



Grading CICE Decompositions

decomposition	cice load balance, ice coverage	cice load balance, radiation	number of neighbors	amount of data to communica te	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. 6: Scorecard

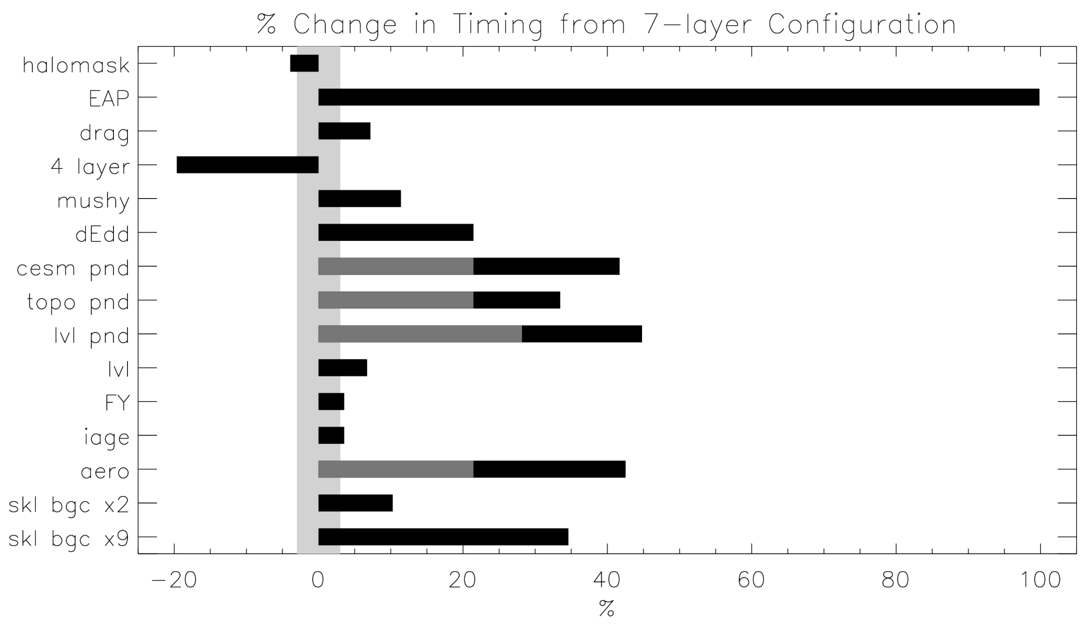


Fig. 7: Timings

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 Time Manager and Initialization

The time manager is an important piece of the CICE model.

Time Manager

The primary prognostic variables in the time manager are `myear`, `mmonth`, `mday`, and `msec`. These are integers and identify the current model year, month, day, and second respectively. The model timestep is `dt` with units of seconds. See *Choosing an appropriate time step* for additional information about choosing an appropriate timestep. The internal variables `istep`, `istep0`, and `istep1` keep track of the number of timesteps. `istep` is the counter for the current run and is set to 0 at the start of each run. `istep0` is the step count at the start of a long multi-restart run, and `istep1` is the step count of a long multi-restart run and is continuous across model restarts.

In general, the time manager should be advanced by calling *advance_timestep*. This subroutine in **ice_calendar.F90** automatically advances the model time by `dt`. It also advances the istep numbers and calls subroutine *calendar* to update additional calendar data.

The namelist variable `use_restart_time` specifies whether to use the time and step numbers saved on a restart file or whether to set the initial model time to the namelist values defined by `year_init`, `month_init`, `day_init`, and `sec_init`. Normally, `use_restart_time` is set to false on the initial run. In continue mode, use_restart_time is ignored and the restart date is always used to initialize the model run. More information about the restart capability can be found in *Restart files*.

Several different calendars are supported including noleap (365 days per year), 360-day (twelve 30 day months per year), and gregorian (leap days every 4 years except every 100 years except every 400 years). The gregorian calendar in CICE is formally a proleptic gregorian calendar without any discontinuities over time. The calendar is set by specifying `days_per_year` and `use_leap_years` in the namelist, and the following combinations are supported,

Table 5: Supported Calendar Options

`days_per_year`	`use_leap_years`	calendar
365	false	noleap
365	true	gregorian
360	false	360-day

The history (*History files*) and restart (*Restart files*) outputs and frequencies are specified in namelist and are computed relative to a reference date defined by the namelist `histfreq_base` and `dumpfreq_base`. Valid values for each are *zero* and *init*. If set to *zero*, all output will be relative to the absolute reference year-month-day date, 0000-01-01. This is the default value for `histfreq_base`, so runs with different initial dates will have identical output. If the `histfreq_base` or `dumpfreq_base` are set to *init*, all frequencies will be relative to the model initial date specified by `year_init`, `month_init`, and `day_init`. `sec_init` plays no role in setting output frequencies. *init* is the default for `dumpfreq_base` and makes it easy to generate restarts 5 or 10 model days after startup as we often do in testing. Both `histfreq_base` and `dumpfreq_base` are arrays and can be set for each stream separately.

In general, output is always written at the start of the year, month, day, or hour without any ability to shift the phase. For instance, monthly output is always written on the first of the month. It is not possible, for instance, to write monthly data once a month on the 10th of the month. In the same way, quarterly data for Dec-Jan-Feb vs Jan-Feb-Mar is not easily controlled. A better approach is to create monthly data and then to aggregate to quarters as a post-processing step. The history and restart (`histfreq`, `dumpfreq`) setting *1* indicates output at a frequency of timesteps. This is the character *1* as opposed to the integer 1. This frequency output is computed using `istep1`, the model timestep. This may vary with each run depending on several factors including the model timestep, initial date, and value of `istep0`.

The model year is limited by some integer math. In particular, calculation of elapsed hours in **ice_calendar.F90**, and the model year is limited to the value of `myear_max` set in that file. Currently, that's 200,000 years.

The time manager was updated in early 2021. The standalone model was modified, and some tests were done in a coupled framework after modifications to the high level coupling interface. For some coupled models, the coupling interface may need to be updated when updating CICE with the new time manager. In particular, the old prognostic variable `time` no longer exists in CICE, `year_init` only defines the model initial year, and the calendar subroutine is called without any arguments. One can set the namelist variables `year_init`, `month_init`, `day_init`, `sec_init`, and `dt` in conjuction with `days_per_year` and `use_leap_years` to initialize the model date, timestep, and calendar. To overwrite the default/namelist settings in the coupling layer, set the **ice_calendar.F90** variables `myear`, `mmonth`, `mday`, `msec` and `dt` after the namelists have been read. Subroutine *calendar* should then be called to update all the calendar data. Finally, subroutine *advance_timestep* should be used to advance the model time manager. It advances the step numbers, advances time by `dt`, and updates the calendar data. The older method of manually advancing the steps and adding `dt` to `time` should be deprecated.

Initialization and Restarts

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 (*Tables 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 *Tables 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 generally control model initialization, `runtype`, `ice_ic`, and `use_restart_time`. The valid values for `runtype` are `initial` or `continue`. When `runtype` = *continue*, the restart filename is stored in a small text (pointer) file, `use_restart_time` is forced to true and `ice_ic` plays no role. When `runtype` = *initial*, `ice_ic` has three options, `none`, `internal`, or *filename*. These initial states are no-ice, namelist driven initial condition, and ice defined by a file respectively. If `ice_ic` is set to `internal`, the initial state is defined by the namelist values `ice_data_type`, `ice_data_dist`, and `ice_data_conc`. In *initial* mode, `use_restart_time` should generally be set to false and the initial time is then defined by `year_init`, `month_init`, `day_init`, and `sec_init`. These combinations options are summarized in *Ice Initialization*.

Restart files and initial condition files are generally the same format and can be the same files. They contain the model state from a particular instance in time. In general, that state includes the physical and dynamical state as well as the state of optional tracers. Reading of various tracer groups can be independently controlled by various restart flags. In other words, a restart file can be used to initialize a new configuration where new tracers are used (i.e. bgc). In that case, the physical state of the model will be read, but if bgc tracers don't exist on the restart file, they can be initialized from scratch.

In `continue` mode, a pointer file is used to restart the model. In this mode, the CICE model writes out a small text (pointer) file to the run directory that names the most recent restart file. On restart, the model reads the pointer file which defines the name of the restart file. The model then reads that restart file. By having this feature, the ice namelist does not need to be constantly updated with the latest restart filename, and the model can be automatically resubmitted. Manually editing the pointer file in the middle of a run will reset the restart filename and allow the run to continue.

Table *Ice Initialization* shows `runtype`, `ice_ic`, and `use_restart_time` namelist combinations for initializing the model. If namelist defines the start date, it's done with `year_init`, `month_init`, `day_init`, and `sec_init`.

Table 6: Ice Initialization

`runtype`	`ice_ic`	`use_restart_time`	Note
initial	*none*	not used	no ice, namelist defines start date
initial	*internal* or *default*	not used	set by namelist ice_data_type, ice_data_dist, ice_data_conc
initial	*filename*	false	read ice state from filename, namelist defines start date
initial	*filename*	true	read ice state from filename, restart file defines start date
continue	not used	not used	pointer file defines restart file, restart file defines start date

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.

An additional namelist option, `restart_coszen` specifies whether the cosine of the zenith angle is included in the restart files. This is mainly used in coupled models.

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.

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, Δt_{dyn} (`dt_dyn`), than the thermodynamics timestep Δt (`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 7: *Time Step Bound*

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

As discussed in [39], 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 [52], limits the time step to $\sim$30 minutes for the old ridging scheme (`krdg_partic` = 0), and to $\sim$2 hours for the new scheme (`krdg_partic` = 1), assuming Δx = 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 ΔH 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 (Δt_e) is thus

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

A second parameter, $E_\circ$ (`elasticDamp`), defines the elastic wave damping timescale T, described in Section *Dynamics*, as `elasticDamp * 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 [21]. Linear stability analysis for the dynamics component shows that the numerical method is stable as long as the subcycling time step Δt_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. The current default parameters for the EVP and EAP are $ndte = 240$ and $E_\circ = 0.36$. For high resolution applications, it is however recommended to increase the value of $ndte$ [30], [5].

Note that only T and Δt_e figure into the stability of the dynamics component; Δt 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 [21]; see [25] 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 [6] (see Section *Revised EVP approach*).

3.1.4 Model Input and Output

IO Overview

CICE provides the ability to read and write binary unformatted or netCDF data via a number of different methods. The IO implementation is specified both at build-time (via selection of specific source code) and run-time (via namelist). Three different IO packages are available in CICE under the directory **cicecore/cicedyn/infrastructure/io**. Those are io_binary, io_netcdf, and io_pio2, and those support IO thru binary, netCDF (https://www.unidata.ucar.edu/software/netcdf), and PIO (https://github.com/NCAR/ParallelIO) interfaces respectively. The io_pio2 directory supports both PIO1 and PIO2 and can write data thru the netCDF or parallel netCDF (pnetCDF) interface. The netCDF history files are CF-compliant, and header information for data contained in the netCDF files is displayed with the command `ncdump -h filename.nc`. To select the io source code, set `ICE_IOTYPE` in **cice.settings** to `binary`, `netcdf`, `pio1`, or `pio2`.

At run-time, more detailed IO settings are available. `restart_format` and `history_format` namelist options specify the method and format further. Valid options are listed in *CICE IO formats*. These options specify the format of new files created by CICE. Existing files can be read in any format as long as it's consistent with `ICE_IOTYPE` defined. Note that with `ICE_IOTYPE = binary`, the format name is actually ignored. The CICE netCDF output contains a global metadata attribute, `io_flavor`, that indicates the format chosen for the file. `ncdump -k filename.nc` also provides information about the specific netCDF file format. In general, the detailed format is not enforced for input files, so any netCDF format can be read in CICE regardless of CICE namelist settings.

Table 8: CICE IO formats

Namelist Option	Format	Written Thru	Valid With ICE_IOTYPE
binary	Fortran binary	fortran	binary
cdf1	netCDF3-classic	netCDF	netcdf, pio1, pio2
cdf2	netCDF3-64bit-offset	netCDF	netcdf, pio1, pio2
cdf5	netCDF3-64bit-data	netCDF	netcdf, pio1, pio2
default	binary or cdf1, depends on ICE_IOTYPE	varies	binary, netcdf, pio1, pio2
hdf5	netCDF4 hdf5	netCDF	netcdf, pio1, pio2
pnetcdf1	netCDF3-classic	pnetCDF	pio1, pio2
pnetcdf2	netCDF3-64bit-offset	pnetCDF	pio1, pio2
pnetcdf5	netCDF3-64bit-data	pnetCDF	pio1, pio2

There are additional namelist options that affect PIO performance for both restart and history output. [`history_`, `restart_`] [`iotasks`,`root`,`stride`] namelist options control the PIO processor/task usage and specify the total number of IO tasks, the root IO task, and the IO task stride respectively. `history_rearranger` and `restart_rearranger` define the PIO rearranger strategy. Finally, [`history_`,`restart_`] [`deflate`,`chunksize`] provide controls for hdf5 compression and chunking for the `hdf5` options in both netCDF and PIO output. `hdf5` is written serially thru the netCDF library and in parallel thru the PIO library in CICE. Additional details about the netCDF and PIO settings and implementations can found in (https://www.unidata.ucar.edu/software/netcdf) and (https://github.com/NCAR/ParallelIO).

netCDF requires CICE compilation with a netCDF library built externally. PIO requires CICE compilation with a PIO and netCDF library built externally. Both netCDF and PIO can be built with many options which may require additional libraries such as MPI, hdf5, or pnetCDF.

History files

CICE provides history data output in binary unformatted or netCDF formats via separate implementations of binary, netCDF, and PIO interfaces as described above. In addition, `history_format` as well as other history namelist options control the specific file format as well as features related to IO performance, see *IO Overview*.

The data is written at the period(s) given by `histfreq` and `histfreq_n` relative to a reference date specified by `histfreq_base`. The files are written to binary or netCDF files prepended by the `history_file` and `history_suffix` namelist setting. The settings for history files are set in the **setup_nml** section of **ice_in** (see *Tables of Namelist Options*). The history filenames will have a form like **[history_file][history_suffix][_freq].[timeID].[nc,da]** depending on the namelist options chosen. 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 *Tables 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 implementation 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_pio2** 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 options (`1`, `h`, `d`, `m`, `y`) are available simultaneously for `histfreq` during a run, and each stream must have a unique value for `histfreq`. In other words, `d` cannot be used by two different streams. Each stream has an associated frequency set by `histfreq_n`. The frequency is relative to a reference date specified by the corresponding entry in `histfreq_base`. Each stream can be instantaneous or time averaged data over the frequency internal. The `hist_avg` namelist turns on time averaging for each stream individually. The same model variable can be written to multiple history streams (ie. daily `d` and monthly `m`) via its namelist flag, $f\_\ \langle var \rangle$, while `x` turns that history variable off. For example, `f_aice = 'md'` will write aice to the monthly and daily streams. Grid variable history output flags are logicals and written to all stream files if turned on. 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 history filenames are set in the subroutine **construct_filename** in **ice_history_shared.F90**. In cases where two streams produce the same identical filename, the model will abort. Use the namelist `hist_suffix` to make stream filenames unique. More information about how the frequency is computed is found in *Time Manager*. Also, some Earth Sytem Models require the history file time axis to be centered in the averaging interval. The flag `hist_time_axis` will allow the user to chose `begin`, `middle`, or `end` for the time stamp.

For example, in the namelist:

```
histfreq   = '1', 'h', 'd', 'm', 'y'
histfreq_n =  1 ,  6 ,  0 ,  1 ,  1
histfreq_base = 'zero','zero','zero','zero','zero'
hist_avg      = .true.,.true.,.true.,.true.,.true.
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. All streams are time averaged over the interval although because one stream has `histfreq=1` and `histfreq_n=1`, that is equivalent to instantaneous output each model timestep.

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`.

There is no restart capability built into the history implementation. If the model stops in the middle of a history accumulation period, that data is lost on restart, and the accumulation is zeroed out at startup. That means the dump frequency (see *Restart files*) and history frequency need to be somewhat coordinated. For example, if monthly history files are requested, the dump frequency should be set to an integer number of months.

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.

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 `hist_avg` averaging flag, at the frequency given by their namelist flag.

The normalized principal components of internal ice stress (`sig1`, `sig2`) 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) [43] 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`.

It may also be helpful for debugging to increase the precision of the history file output from 4 bytes to 8 bytes. This is changed through the `history_precision` namelist flag.

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 defined by the input namelist `lonpnt` and `latpnt`. By default, one point is near the North Pole and the other is in the Weddell Sea; these may be changed in **ice_in**.

The namelist `debug_model` prints detailed debug diagnostics for a single point as the model advances. The point is defined by the namelist `debug_model_i`, `debug_model_j`, `debug_model_iblk`, and `debug_model_task`. These are the local i, j, block, and mpi task index values of the point to be diagnosed. This point is defined in local index space and can be values in the array halo. If the local point is not defined in namelist, the point associated with `lonpnt(1)` and `latpnt(1)` is used. `debug_model` is normally used when the model aborts and needs to be debugged in detail at a particular (usually failing) grid point.

Memory use diagnostics are controlled by the logical namelist `memory_stats`. This feature uses an intrinsic query in C defined in **ice_memusage_gptl.c**. Memory diagnostics will be written at the the frequency defined by diagfreq.

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. The "stats" argument can be set by the `timer_stats` namelist. 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 includes the timings from several other timers, while timer Bound is called from many different places in the code, including the dynamics and advection routines. The Dynamics, Advection, and Column timers do not overlap and represent most of the overall model work.

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

Table 9: CICE timers

Timer		
Index	**Label**	
1	Total	the entire run
2	Timeloop	total minus initialization and exit
3	Dynamics	dynamics
4	Advection	horizontal transport
5	Column	all vertical (column) processes
6	Thermo	vertical thermodynamics, part of Column timer
7	Shortwave	SW radiation and albedo, part of Thermo timer
8	Ridging	mechanical redistribution, part of Column timer
9	FloeSize	flow size, part of Column timer
10	Coupling	sending/receiving coupler messages
11	ReadWrite	reading/writing files
12	Diags	diagnostics (log file)
13	History	history output
14	Bound	boundary conditions and subdomain communications
15	BundBound	halo update bundle copy
16	BGC	biogeochemistry, part of Thermo timer
17	Forcing	forcing
18	1d-evp	1d evp, part of Dynamics timer
19	2d-evp	2d evp, part of Dynamics timer
20	UpdState	update state

Restart files

CICE reads and writes restart data in binary unformatted or netCDF formats via separate implementations of binary, netCDF, and PIO interfaces as described above. In addition, `restart_format` as well as other restart namelist options control the specific file format as well as features related to IO performance, see *IO Overview*.

The restart files created by CICE contain all of the variables needed for a full, exact restart. The filename begins with the character string defined by the `restart_file` namelist input, and the restart dump frequency is given by the namelist variables `dumpfreq` and `dumpfreq_n` relative to a reference date specified by `dumpfreq_base`. Multiple restart frequencies are supported in the code with a similar mechanism to history streams. 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 for initial runs. If `use_restart_time` = false then the initial model date stamp is determined from the namelist parameters, `year_init`, `month_init`, `day_init`, and `sec_init`. 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).

Restart files are available for the CICE code distributions for the gx3 and gx1 grids (see *Forcing data* for information about obtaining these files). They were created using the default model configuration and run for multiple years using the JRA55 forcing.

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

- bash and csh
- gmake (GNU Make)
- Fortran and C compilers (Intel, PGI, GNU, Cray, NVHPC, AOCC, and NAG have been tested)
- NetCDF (optional, but required to test standard configurations that have netCDF grid, input, and forcing files)
- MPI (optional, but required for running on more than 1 processor)
- PIO (optional, but required for running with PIO I/O interfaces)

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. CICE supports both PIO1 and PIO2. To use PIO1, the `USE_PIO1` macro should also be set. 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,

- AOCC 3.0.0
- Intel ifort 15.0.3.187
- Intel ifort 16.0.1.150
- Intel ifort 17.0.1.132
- Intel ifort 17.0.2.174
- Intel ifort 17.0.5.239
- Intel ifort 18.0.1.163
- Intel ifort 18.0.5
- Intel ifort 19.0.2
- Intel ifort 19.0.3.199
- Intel ifort 19.1.0.166
- Intel ifort 19.1.1.217
- Intel ifort 19.1.2.254
- Intel ifort 2021.4.0
- Intel ifort 2021.6.0
- Intel ifort 2021.8.0
- Intel ifort 2021.9.0
- Intel ifort 2022.2.1
- PGI 16.10.0

- PGI 19.9-0
- PGI 20.1-0
- PGI 20.4-0
- GNU 6.3.0
- GNU 7.2.0
- GNU 7.3.0
- GNU 7.7.0
- GNU 8.3.0
- GNU 9.3.0
- GNU 10.1.0
- GNU 11.2.0
- GNU 12.1.0
- GNU 12.2.0
- Cray CCE 8.5.8
- Cray CCE 8.6.4
- Cray CCE 13.0.2
- Cray CCE 14.0.3
- Cray CCE 15.0.1
- NAG 6.2
- NVC 23.5-0

The Consortium has tested the following MPI implementations and versions,

- MPICH 7.3.2
- MPICH 7.5.3
- MPICH 7.6.2
- MPICH 7.6.3
- MPICH 7.7.0
- MPICH 7.7.6
- MPICH 7.7.7
- MPICH 7.7.19
- MPICH 7.7.20
- MPICH 8.1.14
- MPICH 8.1.21
- MPICH 8.1.25
- Intel MPI 18.0.1
- Intel MPI 18.0.4
- Intel MPI 2019 Update 6

- Intel MPI 2019 Update 8
- MPT 2.14
- MPT 2.17
- MPT 2.18
- MPT 2.19
- MPT 2.20
- MPT 2.21
- MPT 2.22
- MPT 2.25
- mvapich2-2.3.3
- OpenMPI 1.6.5
- OpenMPI 4.0.2

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 4.4.1.1.3
- NetCDF 4.4.1.1.6
- NetCDF 4.4.1.1
- NetCDF 4.4.2
- NetCDF 4.5.0
- NetCDF 4.5.2
- NetCDF 4.6.1.3
- NetCDF 4.6.3
- NetCDF 4.6.3.2
- NetCDF 4.7.2
- NetCDF 4.7.4
- NetCDF 4.8.1
- NetCDF 4.8.1.1
- NetCDF 4.8.1.3
- NetCDF 4.9.0.1
- NetCDF 4.9.0.3
- NetCDF 4.9.2

CICE has been tested with

- PIO 1.10.1

- PIO 2.5.4
- PIO 2.5.9
- PIO 2.6.0
- PIO 2.6.1
- PnetCDF 1.12.2
- PnetCDF 1.12.3
- PnetCDF 2.6.2

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. Use of `--env` is also recommended to specify the compilation environment. `--case` and `--test` modes can use `--set` or `-s` which will turn on various model options. `--test` and `--suite` will require `--testid` to be set and can use `--bdir`, `--bgen`, `--bcmp`, and `--diff` to generate (save) results for regression testing (comparison with prior results). `--tdir` will 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 -e intel
cd mycase
```

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

- **env.[machine]_[env]** 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]_[env]** defines the Makefile macros

- **Makefile** is the makefile used to build the model
- **cice.build** is a script that calls the Makefile 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 submits the **cice.run script** or **cice.test** script. These scripts can also be run interactively or submitted manually without the **cice.submit** script.

Some hints:

- 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
- When the run scripts are submitted, the current **ice_in**, **cice.settings**, and **env.[machine]** files are copied from the case directory into the run directory. Users should generally not edit files in the run directory as these are overwritten when following the standard workflow. **cice.settings** can be sourced to establish the case values in the login shell.
- Some useful aliases can be found in the *Use of Shell Aliases* section
- 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 add user defined preprocessor macros, modify `ICE_CPPDEFS` in **cice.settings** using the syntax `-DCICE_MACRO`.
- 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 subdirectory 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.

cice.setup 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 and is paired with `--env` to define the compilation environment.

`--env`, `-e` ENVIRONMENT1,ENVIRONMENT2,ENVIRONMENT3 specifies the compilation environment 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, different compiler versions, different mpi libraries, or other system settigs. 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.

`--queue` QUEUE
: specifies a batch queue name. This is optional. See *Machine Queue 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 cases and tests (not suites), settings defined in **~/.cice_set** (if it exists) will be included in the –set options. This behaviour can be overridden with the *–ignore-user-set*` command line option.

`--ignore-user-set`
: ignores settings defined in **~/.cice.set** (if it exists) for cases and tests. **~/.cice_set** is always ignored for test suites.

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` and `--queue` are 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 that conflict, then the last option chosen takes precedence. Not all options are compatible with each other.

Settings defined in **~/.cice_set** (if it exists) will be included in the `--set` options. This behaviour can be overridden with the *–ignore-user-set`* command line option. The format of the **~/.cice_set** file is a identical to the `--set` option, a single comma-delimited line of options. Settings on the command line will take precedence over settings defined in **~/.cice_set**.

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`, `boxnodyn`, 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.

More about cice.build

cice.build is copied into the case directory and should be run interactively from the case directory to build the model. CICE is built with make and there is a generic Makefile and a machine specific Macros file in the case directory. **cice.build** is a wrapper for a call to make that includes several other features.

CICE is built as follows. First, the makdep binary is created by compiling a small C program. The makdep binary is then run and dependency files are created. The dependency files are included into the Makefile automatically. As a result, make dependencies do not need to be explicitly defined by the user. In the next step, make compiles the CICE code and generates the cice binary.

The standard and recommended way to run is with no arguments

```
cice.build
```

However, **cice.build** does support a couple other use modes.

```
cice.build [-h|--help]
```

provides a summary of the usage.

```
cice.build [make arguments] [target]
```

turns off most of the features of the cice.build script and turns it into a wrapper for the make call. The arguments and/or target are passed to make and invoked more or less like make [make arguments] [target]. This will be the case if either or both the arguments or target are passed to cice.build. Some examples of that are

```
cice.build --version
```

which will pass –version to make.

```
cice.build targets
```

is a valid target of the CICE Makefile and simply echos all the valid targets of the Makefile.

```
cice.build cice
```

or

```
cice.build all
```

are largely equivalent to running **cice.build** without an argument, although as noted earlier, many of the extra features of the cice.build script are turned off when calling cice.build with a target or an argument. Any of the full builds will compile makdep, generate the source code dependencies, and compile the source code.

```
cice.build [clean|realclean]
cice.build [db_files|db_flags]
cice.build [makdep|depends]
```

are other valid options for cleaning the build, writing out information about the Makefile setup, and building just the makdep tool or the dependency file. It is also possible to target a particular CICE object file.

Finally, there is one important parameter in **cice.settings**. The `ICE_CLEANBUILD` variable defines whether the model is cleaned before a build is carried out. By default, this variable is true which means each invokation of **cice.build** will automatically clean the prior build. If incremental builds are desired to save time during development, the `ICE_CLEANBUILD` setting in **cice.settings** should be modified.

C Preprocessor (CPP) Macros

There are a number of C Preprocessing Macros supported in the CICE model. These allow certain coding features like NetCDF, MPI, or specific Fortran features to be excluded or included during the compile.

The CPPs are defined by the *CPPDEFS* variable in the Makefile. They are defined by passing the -D[CPP] to the C and Fortran compilers (ie. -DUSE_NETCDF) and this is what needs to be set in the *CPPDEFS* variable. The value of *ICE_CPPDEFS* in **cice.settings** is copied into the Makefile *CPPDEFS* variable as are settings hardwired into the **Macros.[machine]_[environment]** file.

In general, `-DFORTRANUNDERSCORE` should always be set to support the Fortran/C interfaces in **ice_shr_reprosum.c**. In addition, if NetCDF is used, `-DUSE_NETCDF` should also be defined. A list of available CPPs can be found in *Table of C Preprocessor (CPP) Macros*.

3.2.3 Porting

There are four basic issues that need to be addressed when porting, and these are addressed in four separate files in the script system,

- setup of the environment such as compilers, environment variables, and other support software (in **env.[machine]_[environment]**)
- setup of the Macros file to support the model build (in **Macros.[machine]_[environment]**)
- setup of the batch submission scripts (in **cice.batch.csh**)
- setup of the model launch command (in **cice.launch.csh**)

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** and **configuration/scripts/cice.launch.csh** files need to be modified. In general, the machine is specified in `cice.setup` with `--mach` and the environment (compiler) is specified with `--env`. mach and env in combination define the compiler, compiler version, supporting libaries, and batch information. Multiple compilation environments can be created for a single machine by choosing unique env names.

- 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, update as needed
- cd .. to **configuration/scripts/**
- Edit the **cice.batch.csh** script to add a section for your machine with batch settings
- Edit the **cice.batch.csh** script to add a section for your machine with 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** and **configuratin/scripts/cice.launch.csh** files, retest, and then add and commit the updated machine files to the repository.

Machine variables

There are several machine specific variables defined in the **env.$[machine]**. These variables are used to generate working cases for a given machine, compiler, and batch system. Some variables are optional.

Table 10: *Machine Settings*

variable	format	description
ICE_MACHINE_MACHNAME	string	machine name
ICE_MACHINE_MACHINFO	string	machine information
ICE_MACHINE_ENVNAME	string	env/compiler name
ICE_MACHINE_ENVINFO	string	env/compiler information
ICE_MACHINE_MAKE	string	make command
ICE_MACHINE_WKDIR	string	root work directory
ICE_MACHINE_INPUTDATA	string	root input data directory
ICE_MACHINE_BASELINE	string	root regression baseline directory
ICE_MACHINE_SUBMIT	string	batch job submission command
ICE_MACHINE_TPNODE	integer	machine maximum MPI tasks per node
ICE_MACHINE_MAXPES	integer	machine maximum total processors per job (optional)
ICE_MACHINE_MAXTHREADS	integer	machine maximum threads per mpi task (optional)
ICE_MACHINE_MAXRUNLENGTH	integer	batch wall time limit in hours (optional)
ICE_MACHINE_ACCT	string	batch default account
ICE_MACHINE_QUEUE	string	batch default queue
ICE_MACHINE_BLDTHRDS	integer	number of threads used during build
ICE_MACHINE_QSTAT	string	batch job status command (optional)
ICE_MACHINE_QUIETMODE	true/false	flag to reduce build output (optional)

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 precedence 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 Porting to Laptops or Personal Computers

To get the required software necessary to build and run CICE, and use the plotting and quality control scripts included in the repository, a conda environment file is available at :

`configuration/scripts/machines/environment.yml`.

This configuration is supported by the Consortium on a best-effort basis on macOS and GNU/Linux. It is untested under Windows, but might work using the Windows Subsystem for Linux.

Once you have installed Miniconda and created the `cice` conda environment by following the procedures in this section, CICE should run on your machine without having to go through the formal *Porting* process outlined above.

Installing Miniconda

We recommend the use of the Miniconda distribution to create a self-contained conda environment from the `environment.yml` file. This process has to be done only once. If you do not have Miniconda or Anaconda installed, you can install Miniconda by following the official instructions, or with these steps:

On macOS:

```
# Download the Miniconda installer to ~/miniconda.sh
curl -L https://repo.anaconda.com/miniconda/Miniconda3-latest-MacOSX-x86_64.sh -o ~/
↪miniconda.sh
# Install Miniconda
bash ~/miniconda.sh

# Follow the prompts

# Close and reopen your shell
```

On GNU/Linux:

```
# Download the Miniconda installer to ~/miniconda.sh
wget https://repo.anaconda.com/miniconda/Miniconda3-latest-Linux-x86_64.sh -O ~/
↪miniconda.sh
# Install Miniconda
bash ~/miniconda.sh

# Follow the prompts
```

(continues on next page)

(continued from previous page)

```
# Close and reopen your shell
```

Note: on some Linux distributions (including Ubuntu and its derivatives), the csh shell that comes with the system is not compatible with conda. You will need to install the tcsh shell (which is backwards compatible with csh), and configure your system to use tcsh as csh:

```
# Install tcsh
sudo apt-get install tcsh
# Configure your system to use tcsh as csh
sudo update-alternatives --set csh /bin/tcsh
```

Initializing your shell for use with conda

We recommend initializing your default shell to use conda. This process has to be done only once.

The Miniconda installer should ask you if you want to do that as part of the installation procedure. If you did not answer "yes", you can use one of the following procedures depending on your default shell. Bash should be your default shell if you are on macOS (10.14 and older) or GNU/Linux.

Note: answering "yes" during the Miniconda installation procedure will only initialize the Bash shell for use with conda.

If your Mac has macOS 10.15 or higher, your default shell is Zsh.

These instructions make sure that the `conda` command is available when you start your shell by modifying your shell's startup file. Also, they make sure not to activate the "base" conda environment when you start your shell. This conda environment is created during the Miniconda installation but is not used for CICE.

For Bash:

```
# Install miniconda as indicated above, then initialize your shell to use conda:
source $HOME/miniconda3/bin/activate
conda init bash

# Don't activate the "base" conda environment on shell startup
conda config --set auto_activate_base false

# Close and reopen your shell
```

For Zsh (Z shell):

```
# Initialize Zsh to use conda
source $HOME/miniconda3/bin/activate
conda init zsh

# Don't activate the "base" conda environment on shell startup
conda config --set auto_activate_base false

# Close and reopen your shell
```

For tcsh:

```
# Install miniconda as indicated above, then initialize your shell to use conda:
source $HOME/miniconda3/etc/profile.d/conda.csh
conda init tcsh

# Don't activate the "base" conda environment on shell startup
conda config --set auto_activate_base false

# Close and reopen your shell
```

For fish:

```
# Install miniconda as indicated above, then initialize your shell to use conda:
source $HOME/miniconda3/etc/fish/conf.d/conda.fish
conda init fish

# Don't activate the "base" conda environment on shell startup
conda config --set auto_activate_base false

# Close and reopen your shell
```

For xonsh:

```
# Install miniconda as indicated above, then initialize your shell to use conda:
source-bash $HOME/miniconda3/bin/activate
conda init xonsh

# Don't activate the "base" conda environment on shell startup
conda config --set auto_activate_base false

# Close and reopen your shell
```

Initializing your shell for conda manually

If you prefer not to modify your shell startup files, you will need to run the appropriate `source` command below (depending on your default shell) before using any conda command, and before compiling and running CICE. These instructions make sure the `conda` command is available for the duration of your shell session.

For Bash and Zsh:

```
# Initialize your shell session to use conda:
source $HOME/miniconda3/bin/activate
```

For tcsh:

```
# Initialize your shell session to use conda:
source $HOME/miniconda3/etc/profile.d/conda.csh
```

For fish:

```
# Initialize your shell session to use conda:
source $HOME/miniconda3/etc/fish/conf.d/conda.fish
```

For xonsh:

```
# Initialize your shell session to use conda:
source-bash $HOME/miniconda3/bin/activate
```

Creating CICE directories and the conda environment

The conda configuration expects some directories and files to be present at `$HOME/cice-dirs`:

```
cd $HOME
mkdir -p cice-dirs/runs cice-dirs/baseline cice-dirs/input
# Download the required forcing from https://github.com/CICE-Consortium/CICE/wiki/CICE-
↪Input-Data
# and untar it at $HOME/cice-dirs/input
```

This step needs to be done only once.

If you prefer that some or all of the CICE directories be located somewhere else, you can create a symlink from your home to another location:

```
# Create the CICE directories at your preferred location
cd ${somewhere}
mkdir -p cice-dirs/runs cice-dirs/baseline cice-dirs/input
# Download the required forcing from https://github.com/CICE-Consortium/CICE/wiki/CICE-
↪Input-Data
# and untar it at cice-dirs/input

# Create a symlink to cice-dirs in your $HOME
cd $HOME
ln -s ${somewhere}/cice-dirs cice-dirs
```

Note: if you wish, you can also create a complete machine port for your computer by leveraging the conda configuration as a starting point. See *Porting*.

Next, create the "cice" conda environment from the `environment.yml` file in the CICE source code repository. You will need to clone CICE to run the following command:

```
conda env create -f configuration/scripts/machines/environment.yml
```

This step needs to be done only once and will maintain a static conda environment. To update the conda environment later, use

```
conda env create -f configuration/scripts/machines/environment.yml --force
```

This will update the conda environment to the latest software versions.

Using the conda configuration

Follow the general instructions in *Overview*, using the `conda` machine name and `macos` or `linux` as compiler names.

On macOS:

```
./cice.setup -m conda -e macos -c ~/cice-dirs/cases/case1
cd ~/cice-dirs/cases/case1
./cice.build
./cice.run
```

On GNU/Linux:

```
./cice.setup -m conda -e linux -c ~/cice-dirs/cases/case1
cd ~/cice-dirs/cases/case1
./cice.build
./cice.run
```

A few notes about the conda configuration:

- This configuration always runs the model interactively, such that `./cice.run` and `./cice.submit` are the same.
- You should not update the packages in the `cice` conda environment, nor install additional packages.
- Depending on the numbers of CPUs in your machine, you might not be able to run with the default MPI configuration (`-p 4x1`). You likely will get an OpenMPI error such as:

  > There are not enough slots available in the system to satisfy the 4 slots that were requested by the application: ./cice

  You can run CICE in serial mode by specifically requesting only one process:

  ```
  ./cice.setup -m conda -e linux -p 1x1 ...
  ```

  If you do want to run with more MPI processes than the number of available CPUs in your machine, you can add the `--oversubscribe` flag to the `mpirun` call in `cice.run`:

  ```
  # For a specific case:
  # Open cice.run and replace the line
  mpirun -np <num> ./cice >&! $ICE_RUNLOG_FILE
  # with
  mpirun -np <num> --oversubscribe ./cice >&! $ICE_RUNLOG_FILE

  # For all future cases:
  # Open configuration/scripts/cice.launch.csh and replace the line
  mpirun -np ${ntasks} ./cice >&! \$ICE_RUNLOG_FILE
  # with
  mpirun -np ${ntasks} --oversubscribe ./cice >&! \$ICE_RUNLOG_FILE
  ```

- It is not recommeded to run other test suites than `quick_suite` or `travis_suite` on a personal computer.
- The conda environment is automatically activated when compiling or running the model using the `./cice.build` and `./cice.run` scripts in the case directory. These scripts source the file `env.conda_{linux.macos}`, which calls `conda activate cice`.
- To use the "cice" conda environment with the Python plotting (see *Timeseries Plotting*) and quality control (QC) scripts (see *Code Validation Testing Procedure*), you must manually activate the environment:

```
cd ~/cice-dirs/cases/case1
conda activate cice
python timeseries.py ~/cice-dirs/cases/case1/logs
conda deactivate  # to deactivate the environment
```

- The environment also contains the Sphinx package necessesary to build the HTML documentation :

```
cd doc
conda activate cice
make html
# Open build/html/index.html in your browser
conda deactivate  # to deactivate the environment
```

3.2.5 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/CICE-Input-Data. 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.

3.2.6 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. It's important to note that when the run scripts are submitted, the current **ice_in**, **cice.settings**, and **env.[machine]** files are copied from the case directory into the run directory. Users should generally not edit files in the run directory as these are overwritten when following the standard workflow.

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

3.2.7 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.

3.2.8 Use of Shell Aliases

This section provides a list of some potentially useful shell aliases that leverage the CICE scripts. These are not defined by CICE and are not required for using CICE. They are provided as an example of what can be done by users. The current **ice_in**, **cice.settings**, and **env.[machine]** files are copied from the case directory into the run directory when the model is run. Users can create aliases leveraging the variables in these files. Aliases like the following can be established in shell startup files or otherwise at users discretion:

```
#!/bin/tcsh
# From a case or run directory, source the necessary environment files to run CICE
alias cice_env 'source env.*; source cice.settings'
# Go from case directory to run directory and back (see https://stackoverflow.com/a/
↪34874698/)
alias cdrun  'set rundir=`\grep "setenv ICE_RUNDIR" cice.settings | awk "{print "\$"NF}
↪"` && cd $rundir'
alias cdcase 'set casedir=`\grep "setenv ICE_CASEDIR" cice.settings | awk "{print "\$"NF}
↪"` && cd $casedir'

#!/bin/bash
# From case/test directory, go to run directory
alias cdrun='cd $(cice_var ICE_RUNDIR)'
# From run directory, go to case/test directory
alias cdcase='cd $(cice_var ICE_CASEDIR)'
# monitor current cice run (from ICE_RUNDIR directory)
alias cice_tail='tail -f $(ls -1t cice.runlog.* |head -1)'
# open log from last CICE run (from ICE_CASEDIR directory)
alias cice_lastrun='$EDITOR $(ls -1t logs/cice.runlog.* |head -1)'
# open log from last CICE build (from ICE_CASEDIR directory)
alias cice_lastbuild='$EDITOR $(ls -1t logs/cice.bldlog.* |head -1)'
# show CICE run directory when run in the case directory
alias cice_rundir='cice_var ICE_RUNDIR'
# open a tcsh shell and source env.* and cice.settings (useful for launching CICE in a␣
↪debugger)
alias cice_shell='tcsh -c "cice_env; tcsh"'

## Functions
# Print the value of a CICE variable ($1) from cice.settings
cice_var() {
\grep "setenv $1" cice.settings | awk "{print "\$"3}"
}
```

3.2.9 Timeseries Plotting

The CICE scripts include two scripts that will generate timeseries figures from a diagnostic output file, a Python version (`timeseries.py`) and a csh version (`timeseries.csh`). Both scripts create the same set of plots, but the Python script has more capabilities, and it's likely that the csh script will be removed in the future.

To use the `timeseries.py` script, the following requirements must be met:

- Python v2.7 or later
- numpy Python package
- matplotlib Python package
- datetime Python package

See *Code Validation Testing Procedure* for additional information about how to setup the Python environment, but we recommend using `pip` as follows:

```
pip install --user numpy
pip install --user matplotlib
pip install --user datetime
```

When creating a case or test via `cice.setup`, the `timeseries.csh` and `timeseries.py` scripts are automatically copied to the case directory. Alternatively, the plotting scripts can be found in `./configuration/scripts`, and can be run from any directory.

The Python script can be passed a directory, a specific log file, or no directory at all:

- If a directory is passed, the script will look either in that directory or in directory/logs for a filename like cice.run*. As such, users can point the script to either a case directory or the `logs` directory directly. The script will use the file with the most recent creation time.
- If a specific file is passed the script parses that file, assuming that the file matches the same form of cice.run* files.
- If nothing is passed, the script will look for log files or a `logs` directory in the directory from where the script was run.

For example:

Run the timeseries script on the desired case.

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

or

```
$ python timeseries.py /p/work1/turner/CICE_RUNS/conrad_intel_smoke_col_1x1_diag1_
↪run1year.t00/logs
```

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

The plotting script will plot the following variables by default, but you can also select specific plots to create via the optional command line arguments.

- 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)

For example, to plot only total ice volume and total snow volume

```
$ python timeseries.py /p/work1/turner/CICE_RUNS/conrad_intel_smoke_col_1x1_diag1_
↪run1year.t00/ --volume --snw_vol
```

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
  python timeseries.py $dir
end
```

Plots are only made for a single output file at a time. The ability to plot output from a series of cice.run* files is not currently possible, but may be added in the future. However, using the `--bdir` option will plot two datasets (from log files) on the same figure.

For the latest help information for the script, run

```
$ python timeseries.py -h
```

The `timeseries.csh` script works basically the same way as the Python version, however it does not include all of the capabilities present in the Python version.

To use the C-Shell version of the script,

```
$ ./timeseries.csh /p/work1/turner/CICE_RUNS/conrad_intel_smoke_col_1x1_diag1_run1year.
↪t00/
```

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 main, 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 or overide the batch account number (`--acct`) and queue name (`--queue`)
- Ability to control how test suites execute (`--setup-only`, `--setup-build`, `--setup-build-run`, `--setup-build-submit`)

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 *cice.setup 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 *cice.setup Command Line Options*)

`--env` ENVIRONMENT1 (see *cice.setup Command Line Options*)

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

`--ignore-user-set` (see *cice.setup Command Line Options*)

`--acct` ACCOUNT (see *cice.setup Command Line Options*)

`--grid` GRID (see *cice.setup Command Line Options*)

`--pes` MxNxBXxBYxMB (see *cice.setup 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 [20]. 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: constant (f=1.46e-4 s^{-1})
- ice_data_type : box2001 (special initial ice mask)
- ice_data_conc : p5
- ice_data_dist : box2001 (special ice concentration initialization)
- atm_data_type : box2001 (special atmospheric and ocean forcing)

Ocean stresses are computed as in [20] 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 [67]. 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 initial ice mask)
- ice_data_conc : c1
- ice_data_dist : uniform

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 testsuite.[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 testsuite.[testid]:

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

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 at the command line have precedence over the test suite 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. Some of the available tests suites are

`quick_suite`
: consists of a handful of basic CICE tests

`base_suite`
: consists of a much large suite of tests covering much of the CICE functionality

`decomp_suite`
: checks that different decompositions and pe counts produce bit-for-bit results

`omp_suite`
: checks that OpenMP single thread and multi-thread cases are bit-for-bit identical

`io_suite`
: tests the various IO options including binary, netcdf, and pio. PIO should be installed locally and accessible to the CICE build system to make full use of this suite.

`perf_suite`
: runs a series of tests to evaluate model scaling and performance

`reprosum_suite`
: verifies that CICE log files are bit-for-bit with different decompositions and pe counts when the bfbflag is set to reprosum

`gridsys_suite`
: tests B, C, and CD grid_ice configurations

`prod_suite`
: consists of a handful of tests running 5 to 10 model years and includes some QC testing. These tests will be relatively expensive and take more time compared to other suites.

`unittest_suite`
: runs unit tests in the CICE repository

`travis_suite`
: consists of a small suite of tests suitable for running on low pe counts. This is the suite used with Github Actions for CI in the workflow.

`first_suite`
: this small suite of tests is redundant with tests in other suites. It runs several of the critical baseline tests that other test compare to. It can improve testing turnaround if listed first in a series of test suites.

When running multiple suites on the command line (i.e. `--suite first_suite,base_suite,omp_suite`) the suites will be run in the order defined by the user and redundant tests across multiple suites will be created and executed only once.

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  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.

`--coverage`
When invoked, code coverage diagnostics are generated. This will modify the build and reduce optimization and generate coverage reports using lcov or codecov tools. General use is not recommended, this is mainly used as a diagnostic to periodically assess test coverage. Please see *Code Coverage Testing* for more information.

`--setup-only`
This is only used by `--suite` and when set, just creates the suite testcases. It does not build or submit them to run. By default, the suites do `--setup-build-submit`.

`--setup-build`
This is only used by `--suite` and when set, just creates and builds the suite testcases. It does not submit them to run. By default, the suites do `--setup-build-submit`.

`--setup-build-run`
This is only used by `--suite` and when set, runs the test cases interactively instead of submitting them in batch. By default, the suites do `--setup-build-submit`.

`--setup-build-submit`
This is only used by `--suite` and when set, sets up the cases, builds them, and submits them. This is the default behavior of suites.

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

As indicated above, **cice.setup** with `--suite` will create a directory called testsuite.[testid]. **cice.setup** also generates a script called **suite.submit** in that directory. **suite.submit** is the script that builds and submits the various test cases in the test suite.

The *cice.setup** options `--setup-only`, `--setup-build`, and `--setup-build-run` modify how **suite.submit** is run by **cice.setup**. **suite.submit** can also be run manually, and the environment variables, SUITE_BUILD (builds the testcases), SUITE_RUN (runs the testcases interactively), and SUITE_SUBMIT (submit the testcases to run) control **suite.submit**. The default values for these variables are

```
SUITE_BUILD = true
SUITE_RUN = false
SUITE_SUBMIT = true
```

which means by default the test suite builds and submits the jobs. By defining other values for those environment variables, users can control the suite script. When using **suite.submit** manually, the string `true` (all lowercase) is the only string that will turn on a feature, and both SUITE_RUN and SUITE_SUBMIT cannot be true at the same time.

By leveraging the **cice.setup** command line arguments `--setup-only`, `--setup-build`, and `--setup-build-run` as well as the environment variables SUITE_BUILD, SUITE_RUN, and SUITE_SUBMIT, users can run **cice.setup** and **suite.submit** in various combinations to quickly setup, setup and build, submit, resubmit, run interactively, or rebuild and resubmit full testsuites quickly and easily. See *Test Suite Examples* for an example.

The script **create_fails.csh** will process the output from results.csh and generate a new test suite file, **fails.ts**, from the failed tests. **fails.ts** can then be edited and passed into `cice.setup --suite fails.ts ...` to rerun subsets of failed tests to more efficiently move thru the development, testing, and validation process. However, a full test suite should be run on the final development version of the code.

To report the test results, as is required for Pull Requests to be accepted into the main 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
```

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 command line options will take precedence.

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
```

(continues on next page)

(continued from previous page)

```
# 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 Validation Test (non bit-for-bit validation)*) Pull Requests to incorporate code into the CICE Consortium main branch. 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_MAIN
```

    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
# 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.

13) **Test suite generation then manual build followed by manual submission**

Specify suite, mach, env, testid.

```
./cice.setup --suite quick_suite,base_suite --mach conrad --env cray,
↪gnu --testid v01a --setup-only
cd testsuite.v01a
setenv SUITE_BUILD true
setenv SUITE_RUN false
setenv SUITE_SUBMIT false
./suite.submit
setenv SUITE_BUILD false
setenv SUITE_RUN false
setenv SUITE_SUBMIT true
./suite.submit
# wait for runs to complete
./results.csh
```

The setenv syntax is for csh/tcsh. In bash, the syntax would be SUITE_BUILD=true.

3.3.3 Unit Testing

Unit testing is supported in the CICE scripts. Unit tests are implemented via a distinct top level driver that tests CICE model features explicitly. These drivers can be found in **cicecore/drivers/unittest/**. In addition, there are some script files that also support the unit testing.

The unit tests build and run very much like the standard CICE model. A case is created and model output is saved to the case logs directory. Unit tests can be run as part of a test suite and the output is compared against an earlier set of output using a simple diff of the log files.

For example, to run the existing calendar unit test as a case,

```
./cice.setup -m onyx -e intel --case calchk01 -p 1x1 -s calchk
cd calchk01
./cice.build
./cice.submit
```

Or to run the existing calendar unit test as a test,

```
./cice.setup -m onyx -e intel --test unittest -p 1x1 --testid cc01 -s calchk --bgen cice.
↪cc01
cd onyx_intel_unittest_gx3_1x1_calchk.cc01/
./cice.build
./cice.submit
```

To create a new unit test, add a new driver in **cicecore/driver/unittest**. The directory name should be the name of the test. Then create the appropriate set_nml or set_env files for the new unittest name in **configuration/scripts/options**. In particular, **ICE_DRVOPT** and **ICE_TARGET** need to be defined in a set_env file. Finally, edit **configuration/scripts/Makefile** and create a target for the unit test. The unit tests calchk or helloworld can be used as examples.

The following strings should be written to the log file at the end of the unit test run. The string "COMPLETED SUCCESSFULLY" will indicate the run ran to completion. The string "TEST COMPLETED SUCCESSFULLY" will indicate all the unit testing passed during the run. The unit test log file output is compared as part of regression testing. The string "RunningUnitTest" indicates the start of the output to compare. That string should be written to the log file at the start of the unit test model output. These strings will be queried by the testing scripts and will impact the test reporting. See other unit tests for examples about how these strings could be written.

The following are brief descriptions of some of the current unit tests,

- **bcstchk** is a unit test that exercises the methods in ice_broadcast.F90. This test does not depend on the CICE grid to carry out the testing. By testing with a serial and mpi configuration, both sets of software are tested independently and correctness is verified.
- **calchk** is a unit test that exercises the CICE calendar over 100,000 years and verifies correctness. This test does not depend on the CICE initialization.
- **gridavgchk** is a unit test that exercises the CICE grid_average_X2Y methods and verifies results.
- **halochk** is a unit test that exercises the CICE haloUpdate methods and verifies results.
- **helloworld** is a simple test that writes out helloworld and uses no CICE infrastructure. This tests exists to demonstrate how to build a unit test by specifying the object files directly in the Makefile
- **optargs** is a unit test that tests passing optional arguments down a calling tree and verifying that the optional attribute is preserved correctly.
- **opticep** is a cice test that turns off the icepack optional arguments passed into icepack. This can only be run with a subset of CICE/Icepack cases to verify the optional arguments are working correctly.

- **sumchk** is a unit test that exercises the methods in ice_global_reductions.F90. This test requires that a CICE grid and decomposition be initialized, so CICE_InitMod.F90 is leveraged to initialize the model prior to running a suite of unit validation tests to verify correctness.

3.3.4 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 main branch 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
```

`report_results.csh` will run `results.csh` by default automatically, but we recommmend running it manually first to verify results before publishing them. `report_results.csh -n` will turn off automatic running of `results.csh`.

The reporting can also be automated in a test suite 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.

3.3.5 Code Coverage Testing

The `--coverage` feature in **cice.setup** provides a method to diagnose code coverage. This argument turns on special compiler flags including reduced optimization and then invokes the gcov tool. Once runs are complete, either lcov or codecov can be used to analyze the results. This option is currently only available with the gnu compiler and on a few systems with modified Macros files. In the current implementation, when `--coverage` is invoked, the sandbox is copied to a new sandbox called something like cice_lcov_yymmdd-hhmmss. The source code in the new sandbox is modified slightly to improve coverage statistics and the full coverage suite is run there.

At the present time, the `--coverage` flag invokes the lcov analysis automatically by running the **report_lcov.csh** script in the test suite directory. The output will show up at the CICE lcov website. To use the tool, you should have write permission for that repository. The lcov tool should be run on a full multi-suite test suite, and it can take several hours to process the data once the test runs are complete. A typical instantiation would be

```
./cice.setup --suite first_suite,base_suite,travis_suite,decomp_suite,reprosum_suite,io_
↪suite,quick_suite --mach cheyenne --env gnu --testid cc01 --coverage
```

Alternatively, codecov analysis can be carried out by manually running the **report_codecov.csh** script from the test suite directory, but there are several ongoing problems with this approach and it is not generally recommended. A script that summarizes the end-to-end process for codecov analysis can be found in ..**configuration/scripts/tests/cice_test_codecov.csh**. The codecov analysis is largely identical to the analysis performed by lcov, codecov just provides a nicer web experience to view the output.

This is a special diagnostic test and is not part of the standard model testing. General use is not recommended, this is mainly used as a diagnostic to periodically assess test coverage.

..Because codecov.io does not support git submodule analysis right now, a customized ..repository has to be created to test CICE with Icepack integrated directly. The repository ..https://github.com/apcraig/Test_CICE_Icepack serves as the current default test repository. ..In general, to setup the code coverage test in CICE, the current CICE main has ..to be copied into the Test_CICE_Icepack repository, then the full test suite ..can be run with the gnu compiler with the `--coverage` argument.

..The test suite will run and then a report will be generated and uploaded to ..the codecov.io site by the ..**report_codecov.csh** script. The env variable CODECOV_TOKEN needs to be defined ..either in the environment or in a file named **~/.codecov_cice_token**. That ..token provides write permission to the Test_CICE_Icepack codecov.io site and is available ..by contacting the Consortium team directly.

..A script that carries out the end-to-end testing can be found in ..**configuration/scripts/tests/cice_test_codecov.csh**

..This is a special diagnostic test and does not constitute proper model testing. ..General use is not recommended, this is mainly used as a diagnostic to periodically ..assess test coverage. The interaction with codecov.io is not always robust and ..can be tricky to manage. Some constraints are that the output generated at runtime ..is copied into the directory where compilation took place. That means each ..test should be compiled separately. Tests that invoke multiple runs ..(such as exact restart and the decomp test) will only save coverage information ..for the last run, so some coverage information may be lost. The gcov tool can ..be a little slow to run on large test suites, and the codecov.io bash uploader ..(that runs gcov and uploads the data to codecov.io) is constantly evolving. ..Finally, gcov requires that the diagnostic output be copied into the git sandbox for ..analysis. These constraints are handled by the current scripts, but may change ..in the future.

3.3.6 Code Validation 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 [24]. The following sections follow [49].

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 μ_d between paired CICE simulations 'a' and 'b' that should be identical. μ_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 [66], 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 [68], 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 [62], which can be justified by analyzing the spectral density of daily samples of ice thickness from 5-year records in CICE Consortium member models [24]. The AR(1) approximation is inadmissible for paired velocity samples, because ice drift possesses periodicity from inertia and tides [18][36][50]. 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. [68] 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 [68], 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 α 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 [68], 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 [68] at the 80% confidence interval generated for $N = 1824$ degrees of freedom and lag-1 autocorrelation r_1.

Table 11: 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 Validation 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 [58]. 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 [58], 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 σ_b is calculated equivalently for run 'b'. In this context, R becomes a weighted correlation coefficient, calculated as

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

given the weighted covariance

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

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 [24].

Code Validation Testing Procedure

The CICE code validation (QC) 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)

QC testing should be carried out using configurations (ie. namelist settings) that exercise the active code modifications. Multiple configurations may need to be tested in some cases. Developers can contact the Consortium for guidance or if there are questions.

In order to generate the files necessary for the validation 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
pip install --user cartopy
```

You can also setup a conda env with the same utitities

```
conda env create -f configuration/scripts/tests/qctest.yml
conda activate qctest
```

To run the validation test, setup a baseline run with the original baseline model and then a perturbation run based on recent model changes. Use `--set 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.

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.

The `cice.t-test.py` script will also attempt to generate plots of the mean ice thickness for both the baseline and test cases. Additionally, if the 2-stage test fails then the script will attempt to plot a map showing the grid cells that failed the test. For a full list of options, run `python cice.t-test.py -h`.

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)
# if you want to replace an existing baseline, you should first delete the directory␣
↪cice.my.baseline in ${ICE_BASELINE}.
# git clone the baseline code

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

# Check the results

cd testsuite.base0
./results.csh
```

(continues on next page)

(continued from previous page)

```
# 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

# Note which tests failed and determine which namelist options are responsible for the␣
↪failures
```

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
# Generate the test case(s) using options or namelist changes to activate new code␣
↪modifications

./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
# modify ice_in to activate the namelist options that were determined above
./cice.build
./cice.submit

# Create the t-test testing data
# From the updated sandbox
# Generate the same test case(s) as the baseline using options or namelist changes to␣
↪activate new code modifications

./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
# modify ice_in to activate the namelist options that were determined above
./cice.build
./cice.submit

# Wait for runs to finish
# Perform the QC test

# From the updated sandbox
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 \
                 /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
```

(continues on next page)

(continued from previous page)

```
INFO:__main__:Quadratic Skill Test Passed for Southern Hemisphere
INFO:__main__:
INFO:__main__:Quality Control Test PASSED
```

3.4 Case Settings, Model Namelist, and CPPs

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. In addition, the first table lists available preprocessor macros to activate or deactivate various features when compiling.

3.4.1 Table of C Preprocessor (CPP) Macros

The CICE model supports a number of C Preprocessor (CPP) Macros. These can be turned on during compilation to activate different pieces of source code. The main purpose is to introduce build-time code modifications to include or exclude certain libraries or Fortran language features. More information can be found in *C Preprocessor (CPP) Macros*. The following CPPs are available.

Table 12: **CPP Macros**

CPP name	description
General Macros	
CESM1_PIO	Provide backwards compatible support for PIO interfaces/version released with CESM1 in about 2010
ESMF_INTERFACE	Turns on ESMF support in a subset of driver code. Also USE_ESMF_LIB and USE_ESMF_METADATA
FORTRANUNDERSCORE	Used in ice_shr_reprosum86.c to support Fortran-C interfaces. This should generally be turned on at all times. There are other CPPs (FORTRANDOUBULEUNDERSCORE, FORTRANCAPS, etc) in ice_shr_reprosum.c that are generally not used in CICE but could be useful if problems arise in the Fortran-C interfaces
GPTL	Turns on GPTL initialization if needed for PIO
NO_F2003	Turns off some Fortran 2003 features
NO_I8	Converts integer*8 to integer*4. This could have adverse affects for certain algorithms including the ddpdd implementation associated with the `bfbflag`
NO_R16	Converts real*16 to real*8. This could have adverse affects for certain algorithms including the lsum16 implementation associated with the `bfbflag`
NO_SNICARHC	Does not compile hardcoded (HC) 5 band snicar tables tables needed by `shortwave=dEdd_snicar_ad`. May reduce compile time.
USE_NETCDF	Turns on netCDF code. This is normally on and is needed for released configurations. An older value, ncdf, is still supported.
USE_PIO1	Modifies CICE PIO implementation to be compatible with PIO1. By default, code is compatible with PIO2
Application Macros	
CESMCOUPLED	Turns on code changes for the CESM coupled application
CICE_IN_NEMO	Turns on code changes for coupling in the NEMO ocean model
CICE_DMI	Turns on code changes for the DMI coupled model application
ICE_DA	Turns on code changes in the hadgem driver
RASM_MODS	Turns on code changes for the RASM coupled application
Library Macros	
_OPENMP	Automatically defined when compiling with OpenMP
_OPENACC	Automatically defined when compiling with OpenACC

3.4.2 Table of CICE Settings

The **cice.settings** file contains a number of environment variables that define configuration, file system, run, and build settings. Several variables are set by the **cice.setup** script. This file is created on a case by case basis and can be modified as needed.

Table 13: **CICE settings**

variable	options/format	description	default 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_ENVNAME	string	environment name	set by cice.setup

continues on next page

Table 13 – continued from previous page

variable	options/format	description	default value
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	standalone/cice
ICE_TARGET	string	build target	set by cice.setup
ICE_IOTYPE	string	I/O source code	set by cice.setup
	binary	uses io_binary directory, no support for netCDF files	
	netcdf	uses io_netCDF directory, supports netCDF files	
	pio1	uses io_pio directory with PIO1 library, supports netCDF and parallel netCDF thru PIO interfaces	
	pio2	uses io_pio directory with PIO2 library, supports netCDF and parallel netCDF thru PIO interfaces	
ICE_CLEANBUILD	true, false	automatically clean before building	true
ICE_CPPDEFS	user defined preprocessor macros for build	null	
ICE_QUIETMODE	true, false	reduce build output to the screen	false
ICE_GRID	string (see below)	grid	set by cice.setup
	gbox12	12x12 box	
	gbox80	80x80 box	
	gbox128	128x128 box	
	gbox180	180x180 box	
	gx1	1-deg displace-pole (Greenland) global grid	
	gx3	3-deg displace-pole (Greenland) global grid	
	tx1	1-deg tripole global grid	
ICE_NTASKS	integer	number of MPI tasks	set by cice.setup
ICE_NTHRDS	integer	number of threads per task	set by cice.setup
ICE_OMPSCHED	string	OpenMP SCHEDULE env setting	static,1
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_TESTID	string	test name testid	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
ICE_BASECOM	string	baseline directory name for regression comparison, associated with cice.setup -bcmp	set by cice.setup

continues on next page

Table 13 – continued from previous page

variable	options/format	description	default value
ICE_BFBCOMP	string	location of case for comparison, associated with cice.setup –bcmp	set by cice.setup
ICE_BFBTYPE	string	type and files used in BFBCOMP	restart
	log	log file comparison for bit for bit	
	logrest	log and restart files for bit for bit	
	qcchk	QC test for same climate	
	qcchkf	QC test for different climate	
	restart	restart files for bit for bit	
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
	-1	15 minutes (default)	
	0	30 minutes	
	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_COMMDIR	mpi, serial	specify infrastructure comm version	set by ICE_NTASKS
ICE_SNICARHC	true, false	turn on hardcoded (HC) SNICAR tables in Icepack	false
ICE_BLDDEBUG	true, false	turn on compile debug flags	false
ICE_COVERAGE	true, false	turn on code coverage flags	false

3.4.3 Tables of Namelist Options

CICE reads a namelist input file, **ice_in**, consisting of several namelist groups. The tables below summarize the different groups and the variables in each group. The variables are organized alphabetically and the default values listed are the values defined in the source code. Those values will be used unless overridden by the CICE namelist file, **ice_in**. The source code default values as listed in the table are not necessarily the recommended production values.

setup_nml

Table 14: **setup_nml namelist options**

variable	options/format	description	default value
`bfbflag`	`off`	local reduction then global scalar sum	`off`
	`lsum4`	local reduction with real*4 then global scalar sum	
	`lsum8`	local reduction with real*8 then global scalar sum	

continues on next page

Table 14 – continued from previous page

variable	options/format	description	default value
	lsum16	local reduction with real*16 then global scalar sum	
	ddpdd	parallel double double algorithm	
	reprosum	fixed point double integer sum	
conserv_check	logical	check conservation	.false.
cpl_bgc	logical	couple bgc thru driver	.false.
days_per_year	integer	number of days in a model year	365
day_init	integer	the initial day of the month if not using restart	1
debug_forcing	logical	write extra forcing diagnostics	.false.
debug_model	logical	write extended model point diagnostics	.false.
debug_model_i	integer	local i index of debug_model point	-1
debug_model_iblk	integer	iblk value for debug_model point	-1
debug_model_j	integer	local j index of debug_model point	-1
debug_model_task	integer	mpi task value for debug_model point	-1
debug_model_step	logical	initial timestep to write debug_model output	0
diagfreq	integer	frequency of diagnostic output in timesteps	24
diag_type	stdout	write diagnostic output to stdout	stdout
	file	write diagnostic output to file	
diag_file	string	diagnostic output file	'ice_diag.d'
dt	real	thermodynamics time step length in seconds	3600.
dumpfreq	d	write restart every dumpfreq_n days	'y','x','x','x','x'
	d1	write restart once after dumpfreq_n days	
	h	write restart every dumpfreq_n hours	
	h1	write restart once after dumpfreq_n hours	
	m	write restart every dumpfreq_n months	
	m1	write restart once after dumpfreq_n months	
	y	write restart every dumpfreq_n years	
	y1	write restart once after dumpfreq_n years	
	1	write restart every dumpfreq_n time steps	
	11	write restart once after dumpfreq_n time steps	
dumpfreq_base	init	restart output frequency relative to year_init, month_init, day_init	'init','init','init','init','init'
	zero	restart output frequency relative to year-month-day of 0000-01-01	
dumpfreq_n	integer array	write restart frequency with dumpfreq	1,1,1,1,1
dump_last	logical	write restart on last time step of simulation	.false.
histfreq	d	write history every histfreq_n days	'1','h','d','m','y'
	h	write history every histfreq_n hours	
	m	write history every histfreq_n months	
	x	unused frequency stream (not written)	
	y	write history every histfreq_n years	
	1	write history every histfreq_n time step	
histfreq_base	init	history output frequency relative to year_init, month_init, day_init	'zero','zero','zero','zero','zero'

continues on next page

Table 14 – continued from previous page

variable	options/format	description	default value
	zero	history output frequency relative to year-month-day of 0000-01-01	
`histfreq_n`	integer array	frequency history output is written with `histfreq`	1,1,1,1,1
`history_chunksize`	integer array	chunksizes (x,y) for history output (hdf5 only)	0,0
`history_deflate`	integer	compression level (0 to 9) for history output (hdf5 only)	0
`history_dir`	string	path to history output directory	'./'
`history_file`	string	output file for history	'iceh'
`history_format`	`binary`	write history files with binary format	`cdf1`
	`cdf1`	write history files with netcdf cdf1 (netcdf3-classic) format	
	`cdf2`	write history files with netcdf cdf2 (netcdf3-64bit-offset) format	
	`cdf5`	write history files with netcdf cdf5 (netcdf3-64bit-data) format	
	`default`	write history files in default format	
	`hdf5`	write history files with netcdf hdf5 (netcdf4) format	
	`pio_pnetcdf`	write history files with pnetcdf in PIO, deprecated	
	`pio_netcdf`	write history files with netcdf in PIO, deprecated	
	`pnetcdf1`	write history files with pnetcdf cdf1 (netcdf3-classic) format	
	`pnetcdf2`	write history files with pnetcdf cdf2 (netcdf3-64bit-offset) format	
	`pnetcdf5`	write history files with pnetcdf cdf5 (netcdf3-64bit-data) format	
`history_iotasks`	integer	pe io tasks for history output with history_root and history_stride (PIO only), -99=internal default	-99
`history_precision`	integer	history file precision: 4 or 8 byte	4
`history_rearranger`	box	box io rearranger option for history output (PIO only)	default
	default	internal default io rearranger option for history output	
	subset	subset io rearranger option for history output	
`history_root`	integer	pe root task for history output with history_iotasks and history_stride (PIO only), -99=internal default	-99
`history_stride`	integer	pe stride for history output with history_iotasks and history_root (PIO only), -99=internal default	-99
`hist_avg`	logical	write time-averaged data	`.true.,.true.,.true.,.true.,.true.`
`hist_suffix`	character array	appended to history_file when not x	`x,x,x,x,x`

continues on next page

Table 14 – continued from previous page

variable	options/format	description	default value
`hist_time_axis`	character	history file time axis interval location: begin, middle, end	end
`ice_ic`	`default`	equal to internal	`default`
	`internal`	initial conditions set based on ice_data_type,conc,dist inputs	
	`none`	no ice	
	'path/file'	restart file name	
`incond_dir`	string	path to initial condition directory	'./'
`incond_file`	string	output file prefix for initial condition	'iceh_ic'
`istep0`	integer	initial time step number	0
`latpnt`	real	latitude of (2) diagnostic points	90.0,-65.0
`lcdf64`	logical	use 64-bit netCDF format, deprecated, see history_format, restart_format	`.false.`
`lonpnt`	real	longitude of (2) diagnostic points	0.0,-45.0
`memory_stats`	logical	turns on memory use diagnostics	`.false.`
`month_init`	integer	the initial month if not using restart	1
`ndtd`	integer	number of dynamics/advection/ridging/steps per thermo timestep	1
`npt`	integer	total number of npt_units to run the model	99999
`npt_unit`	`d`	run `npt` days	1
	`h`	run `npt` hours	
	`m`	run `npt` months	
	`s`	run `npt` seconds	
	`y`	run `npt` years	
	`1`	run `npt` timesteps	
`numin`	integer	minimum internal IO unit number	11
`numax`	integer	maximum internal IO unit number	99
`pointer_file`	string	restart pointer filename	'ice.restart_file'
`print_global`	logical	print global sums diagnostic data	`.true.`
`print_points`	logical	print diagnostic data for two grid points	`.false.`
`restart`	logical	exists but deprecated, now set internally based on other inputs	
`restart_chunksize`	integer array	chunksizes (x,y) for restart output (hdf5 only)	0,0
`restart_deflate`	integer	compression level (0 to 9) for restart output (hdf5 only)	0
`restart_dir`	string	path to restart directory	'./'
`restart_ext`	logical	read/write halo cells in restart files	`.false.`
`restart_file`	string	output file prefix for restart dump	'iced'
`restart_format`	`binary`	write restart files with binary format	`cdf1`
	`cdf1`	write restart files with netcdf cdf1 (netcdf3-classic) format	
	`cdf2`	write restart files with netcdf cdf2 (netcdf3-64bit-offset) format	
	`cdf5`	write restart files with netcdf cdf5 (netcdf3-64bit-data) format	
	`default`	write restart files in default format	
	`hdf5`	write restart files with netcdf hdf5 (netcdf4) format	

continues on next page

Table 14 – continued from previous page

variable	options/format	description	default value
	`pio_pnetcdf`	write restart files with pnetcdf in PIO, deprecated	
	`pio_netcdf`	write restart files with netcdf in PIO, deprecated	
	`pnetcdf1`	write restart files with pnetcdf cdf1 (netcdf3-classic) format	
	`pnetcdf2`	write restart files with pnetcdf cdf2 (netcdf3-64bit-offset) format	
	`pnetcdf5`	write restart files with pnetcdf cdf5 (netcdf3-64bit-data) format	
`restart_iotasks`	integer	pe io tasks for restart output with restart_root and restart_stride (PIO only), -99=internal default	-99
`restart_rearranger`	box	box io rearranger option for restart output (PIO only)	default
	default	internal default io rearranger option for restart output	
	subset	subset io rearranger option for restart output	
`restart_root`	integer	pe root task for restart output with restart_iotasks and restart_stride (PIO only), -99=internal default	-99
`restart_stride`	integer	pe stride for restart output with restart_iotasks and restart_root (PIO only), -99=internal default	-99
`runid`	string	label for run (currently CESM only)	'unknown'
`runtype`	`continue`	restart using `pointer_file`	`initial`
	`initial`	start from `ice_ic`	
`sec_init`	integer	the initial second if not using restart	0
`timer_stats`	logical	controls extra timer output	`.false.`
`use_leap_years`	logical	include leap days	`.false.`
`use_restart_time`	logical	set initial date using restart file on initial runtype only	`.false.`
`version_name`	string	model version	'unknown_version_name'
`write_ic`	logical	write initial condition	`.false.`
`year_init`	integer	the initial year if not using restart	0

grid_nml

Table 15: **grid_nml namelist options**

variable	options/format	description	default value
`bathymetry_file`	string	name of bathymetry file to be read	'unknown_bathymetry_file'
`bathymetry_format`	`default`	NetCDF depth field	'default'
	`pop`	pop thickness file in cm in ascii format	

continues on next page

Table 15 – continued from previous page

variable	options/format	description	default value
`close_boundaries`	logical	force two gridcell wide land mask on boundaries for rectangular grids	`.false.`
`dxrect`	real	x-direction grid spacing for rectangular grid in cm	0.0
`dxscale`	real	user defined rectgrid x-grid scale factor	1.0
`dyrect`	real	y-direction grid spacing for rectangular grid in cm	0.0
`dyscale`	real	user defined rectgrid y-grid scale factor	1.0
`gridcpl_file`	string	input file for coupling grid info	'unknown_gridcpl_file'
`grid_atm`	`A`	atm forcing/coupling grid, all fields on T grid	`A`
	`B`	atm forcing/coupling grid, thermo fields on T grid, dyn fields on U grid	
	`C`	atm forcing/coupling grid, thermo fields on T grid, dynu fields on E grid, dynv fields on N grid	
	`CD`	atm forcing/coupling grid, thermo fields on T grid, dyn fields on N and E grid	
`grid_file`	string	name of grid file to be read	'unknown_grid_file'
`grid_format`	`bin`	read direct access grid and kmt files	`bin`
	`nc`	read grid and kmt files	
`grid_ice`	`B`	use B grid structure with T at center and U at NE corner	`B`
	`C`	use C grid structure with T at center, U at E edge, V at N edge	
`grid_ocn`	`A`	ocn forcing/coupling grid, all fields on T grid	`A`
	`B`	ocn forcing/coupling grid, thermo fields on T grid, dyn fields on U grid	
	`C`	ocn forcing/coupling grid, thermo fields on T grid, dynu fields on E grid, dynv fields on N grid	
	`CD`	ocn forcing/coupling grid, thermo fields on T grid, dyn fields on N and E grid	
`grid_type`	`displaced_pole`	read from file in *popgrid*	`rectangular`
	`rectangular`	defined in *rectgrid*	
	`regional`	read from file in *popgrid*	
	`tripole`	read from file in *popgrid*	
`kcatbound`	`-1`	single category formulation	1
	`0`	old formulation	
	`1`	new formulation with round numbers	
	`2`	WMO standard categories	
	`3`	asymptotic scheme	
`kmt_file`	string	name of land mask file to be read	`unknown_kmt_file`
`kmt_type`	boxislands	ocean/land mask set internally, complex test geometory	file
	channel	ocean/land mask set internally as zonal channel	

continues on next page

Table 15 – continued from previous page

variable	options/format	description	default value
	channel_oneeast	ocean/land mask set internally as single gridcell east-west zonal channel	
	channel_onenorth	ocean/land mask set internally as single gridcell north-south zonal channel	
	default	ocean/land mask set internally, land in upper left and lower right of domain,	
	file	ocean/land mask setup read from file, see kmt_file	
	wall	ocean/land mask set at right edge of domain	
`latrefrect`	real	lower left corner lat for rectgrid in deg	71.35
`lonrefrect`	real	lower left corner lon for rectgrid in deg	-156.5
`nblyr`	integer	number of zbgc layers	0
`ncat`	integer	number of ice thickness categories	0
`nfsd`	integer	number of floe size categories	1
`nilyr`	integer	number of vertical layers in ice	0
`nslyr`	integer	number of vertical layers in snow	0
`orca_halogrid`	logical	use orca haloed grid for data/grid read	`.false.`
`scale_dxdy`	logical	apply dxscale, dyscale to rectgrid	`false`
`use_bathymetry`	logical	use read in bathymetry file for seabedstress option	`.false.`

domain_nml

Table 16: **domain_nml namelist options**

variable	options/format	description	default value
`add_mpi_barriers`	logical	throttle communication	`.false.`
`block_size_x`	integer	block size in x direction	-1
`block_size_y`	integer	block size in y direction	-1
`debug_blocks`	logical	add additional print statements to debug the block decomposition	`.false.`
`distribution_type`	`cartesian`	2D cartesian block distribution method	`cartesian`
	`rake`	redistribute blocks among neighbors	
	`roundrobin`	1 block per proc until blocks are used	
	`sectcart`	blocks distributed to domain quadrants	
	`sectrobin`	several blocks per proc until used	
	`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 weight method with land block elimination	`latitude`
	`blockall`	full block weight method without land block elimination	
	`latitude`	latitude/ocean sets `work_per_block`	

continues on next page

Table 16 – continued from previous page

variable	options/format	description	default value
`distribution_wght_file`	string	distribution weight file when distribution_type is `wghtfile`	'unknown'
`ew_boundary_type`	`cyclic`	periodic boundary conditions in x-direction	`cyclic`
	`open`	Dirichlet boundary conditions in x	
`maskhalo_dyn`	logical	mask unused halo cells for dynamics	`.false.`
`maskhalo_remap`	logical	mask unused halo cells for transport	`.false.`
`maskhalo_bound`	logical	mask unused halo cells for boundary updates	`.false.`
`max_blocks`	integer	maximum number of blocks per MPI task for memory allocation	-1
`nprocs`	integer	number of processors to use	-1
`ns_boundary_type`	`cyclic`	periodic boundary conditions in y-direction	`open`
	`open`	Dirichlet boundary conditions in y	
	`tripole`	U-fold tripole boundary conditions in y	
	`tripoleT`	T-fold tripole boundary conditions in y	
`nx_global`	integer	global grid size in x direction	-1
`ny_global`	integer	global grid size in y direction	-1
`processor_shape`	`slenderX1`	1 processor in the y direction used with `distribution_type=cartesian`	`slenderX2`
	`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, ∼ square	
	`square-pop`	more processors in y than x, ∼ square	

tracer_nml

Table 17: **tracer_nml namelist options**

variable	options/format	description	default value
`n_aero`	integer	number of aerosol tracers	0
`n_algae`	0,1,2,3	number of algal tracers	0
`n_dic`	0,1	number of dissolved inorganic carbon	0
`n_doc`	0,1,2,3	number of dissolved organic 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
`n_iso`	integer	number of isotope tracers	0
`n_zaero`	0,1,2,3,4,5,6	number of z aerosol tracers in use	0
`tr_aero`	logical	aerosols	`.false.`
`tr_fsd`	logical	floe size distribution	`.false.`
`tr_FY`	logical	first-year ice area	`.false.`
`tr_iage`	logical	ice age	`.false.`
`tr_iso`	logical	isotopes	`.false.`
`tr_lvl`	logical	level ice area and volume	`.false.`
`tr_pond_lvl`	logical	level-ice melt ponds	`.false.`
`tr_pond_cesm`		DEPRECATED	
`tr_pond_topo`	logical	topo melt ponds	`.false.`
`tr_snow`	logical	advanced snow physics	`.false.`

continues on next page

Table 17 – continued from previous page

variable	options/format	description	default value
restart_aero	logical	restart tracer values from file	.false.
restart_age	logical	restart tracer values from file	.false.
restart_fsd	logical	restart floe size distribution values from file	.false.
restart_FY	logical	restart tracer values from file	.false.
restart_iso	logical	restart tracer values from file	.false.
restart_lvl	logical	restart tracer values from file	.false.
restart_pond_lvl	logical	restart tracer values from file	.false.
restart_pond_topo	logical	restart tracer values from file	.false.
restart_snow	logical	restart snow tracer values from file	.false.

thermo_nml

Table 18: **thermo_nml namelist options**

variable	options/format	description	default value
a_rapid_mode	real	brine channel diameter in m	0.5e-3
aspect_rapid_mode	real	brine convection aspect ratio	1.0
conduct	bubbly	conductivity scheme [45]	bubbly
	MU71	conductivity [40]	
dSdt_slow_mode	real	slow drainage strength parameter m/s/K	-1.5e-7
floediam	real	effective floe diameter for lateral melt in m	300.0
hfrazilmin	real	min thickness of new frazil ice in m	0.05
hi_min	real	minimum ice thickness in m	0.01
kitd	0	delta function ITD approximation	1
	1	linear remapping ITD approximation	
ksno	real	snow thermal conductivity	0.3
ktherm	-1	thermodynamic model disabled	1
	1	Bitz and Lipscomb thermodynamic model	
	2	mushy-layer thermodynamic model	
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
Rac_rapid_mode	real	critical Rayleigh number	10.0
Tliquidus_max	real	maximum liquidus temperature of mush (C)	0.0

dynamics_nml

Table 19: **dynamics_nml namelist options**

variable	options/format	description	default value
advection	remap	linear remapping advection scheme	remap
	upwind	donor cell advection	
algo_nonlin	anderson	use nonlinear anderson algorithm for implicit solver	picard
	picard	use picard algorithm	

continues on next page

Table 19 – continued from previous page

variable	options/format	description	default value
alphab	real	α_b factor in [34]	20.0
arlx	real	revised_evp value	300.0
brlx	real	revised_evp value	300.0
capping_method	max	max capping in [15]	max
	sum	sum capping in [31]	
Cf	real	ratio of ridging work to PE change in ridging	17.0
coriolis	constant	constant coriolis value = 1.46e-4 s^{-1}	latitude
	latitude	coriolis variable by latitude	
	zero	zero coriolis	
Cstar	real	constant in Hibler strength formula	20
deltaminEVP	real	minimum delta for viscosities	1e-11
deltaminVP	real	minimum delta for viscosities	2e-9
dim_fgmres	integer	maximum number of Arnoldi iterations for FGMRES solver	50
dim_pgmres	integer	maximum number of Arnoldi iterations for PGMRES preconditioner	5
e_plasticpot	real	aspect ratio of elliptical plastic potential	2.0
e_yieldcurve	real	aspect ratio of elliptical yield curve	2.0
elasticDamp	real	elastic damping parameter	0.36
evp_algorithm	standard_2d	standard 2d EVP memory parallel solver	standard_2d
	shared_mem_1d	1d shared memory solver	
kdyn	-1	dynamics algorithm OFF	1
	0	dynamics OFF	
	1	EVP dynamics	
	2	EAP dynamics	
	3	VP dynamics	
kstrength	0	ice strength formulation [15]	1
	1	ice strength formulation [52]	
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	
kridge	-1	ridging disabled	1
	1	ridging enabled	
ktransport	-1	transport disabled	1
	1	transport enabled	
Ktens	real	Tensile strength factor (see [3])	0.0
k1	real	1st free parameter for landfast parameterization	7.5
k2	real	2nd free parameter (N/m^3) for landfast parameterization	15.0
maxits_fgmres	integer	maximum number of restarts for FGMRES solver	1
maxits_nonlin	integer	maximum number of nonlinear iterations for VP solver	10
maxits_pgmres	integer	maximum number of restarts for PGMRES preconditioner	1
monitor_fgmres	logical	write velocity norm at each FGMRES iteration	.false.

continues on next page

Table 19 – continued from previous page

variable	options/format	description	default value
monitor_nonlin	logical	write velocity norm at each nonlinear iteration	.false.
monitor_pgmres	logical	write velocity norm at each PGMRES iteration	.false.
mu_rdg	real	e-folding scale of ridged ice for krdg_partic = 1 in m^0.5	3.0
ndte	integer	number of EVP subcycles	120
ortho_type	cgs	Use classical Gram-Shchmidt in FGMRES solver	mgs
	mgs	Use modified Gram-Shchmidt in FGMRES solver	
precond	diag	Use Jacobi preconditioner for the FGMRES solver	pgmres
	ident	Don't use a preconditioner for the FGMRES solver	
	pgmres	Use GMRES as preconditioner for FGMRES solver	
Pstar	real	constant in Hibler strength formula (N/m^2)	2.75e4
reltol_fgmres	real	relative tolerance for FGMRES solver	1e-1
reltol_nonlin	real	relative tolerance for nonlinear solver	1e-8
reltol_pgmres	real	relative tolerance for PGMRES preconditioner	1e-6
revised_evp	logical	use revised EVP formulation	.false.
seabed_stress	logical	use seabed stress parameterization for landfast ice	.false.
seabed_stress_method	LKD	linear keel draft method [34]	LKD
	probabilistic	probability of contact method [11]	
ssh_stress	coupled	computed from coupled sea surface height gradient	geostrophic
	geostropic	computed from ocean velocity	
threshold_hw	real	Max water depth for grounding (see [1])	30.
use_mean_vrel	logical	Use mean of two previous iterations for vrel in VP	.true.
visc_method	avg_strength	average strength for viscosities on U grid	avg_zeta
	avg_zeta	average zeta for viscosities on U grid	
yield_curve	ellipse	elliptical yield curve	ellipse

shortwave_nml

Table 20: **shortwave_nml namelist options**

variable	options/format	description	default value
`ahmax`	real	albedo is constant above this thickness in meters	0.3
`albedo_type`	*ccsm3*`	NCAR CCSM3 albedo implementation	`ccsm3`
	`constant`	four constant albedos	
`albicei`	$0 < \alpha < 1$	near infrared ice albedo for thicker ice	0.36
`albicev`	$0 < \alpha < 1$	visible ice albedo for thicker ice	0.78
`albsnowi`	$0 < \alpha < 1$	near infrared, cold snow albedo	0.70
`albsnowv`	$0 < \alpha < 1$	visible, cold snow albedo	0.98
`dT_mlt`	real	Δ temperature per Δ snow grain radius	1.5
`kalg`	real	absorption coefficient for algae	0.6
`rsnw_mlt`	real	maximum melting snow grain radius	1500.
`R_ice`	real	tuning parameter for sea ice albedo from Delta-Eddington shortwave	0.0
`R_pnd`	real	tuning parameter for ponded sea ice albedo from Delta-Eddington shortwave	0.0
`R_snw`	real	tuning parameter for snow (broadband albedo) from Delta-Eddington shortwave	1.5
`shortwave`	`ccsm3`	NCAR CCSM3 shortwave distribution method	`ccsm3`
	`dEdd`	Delta-Eddington method (3-band)	
	`dEdd_snicar_ad`	Delta-Eddington method with 5 band snow	
`snw_ssp_table`	`snicar`	lookup table for *dEdd_snicar_ad*	`test`
	`test`	reduced lookup table for *dEdd_snicar_ad* testing	
`sw_dtemp`	real	temperature difference from melt to start redistributing	0.02
`sw_frac`	real	fraction redistributed	0.9
`sw_redist`	logical	redistribute internal shortwave to surface	`.false.`

ponds_nml

Table 21: **ponds_nml namelist options**

variable	options/format	description	default value
`dpscale`	real	time scale for flushing in permeable ice	1.0
`frzpnd`	`cesm`	CESM pond refreezing forumulation	`cesm`
	`hlid`	Stefan refreezing with pond ice thickness	
`hp1`	real	critical ice lid thickness for topo ponds in m	0.01
`hs0`	real	snow depth of transition to bare sea ice in m	
`hs1`	real	snow depth of transition to pond ice in m	0.03
`pndaspect`	real	aspect ratio of pond changes (depth:area)	0.8
`rfracmax`	$0 \leq r_{max} \leq 1$	maximum melt water added to ponds	0.85
`rfracmin`	$0 \leq r_{min} \leq 1$	minimum melt water added to ponds	0.15

snow_nml

Table 22: **snow_nml namelist options**

variable	options/format	description	default value
`drhosdwind`	real	wind compactions factor for now in kg-s/m^4	27.3
`rhosmax`	real	maximum snow density in kg/m^3	450.
`rhosmin`	real	minimum snow density in kg/m^3	100.
`rhosnew`	real	new snow density in kg/m^3	100.
`rsnw_fall`	real	radius of new snow in 1.0e-6 m	100.
`rsnw_tmax`	real	maximum snow radius in 1.0e-6 m	1500.
`snwgrain`	logical	snow metamorophsis flag	`.false.`
`snwlvlfac`	real	fractional increase in snow	0.3
`snwredist`	`bulk`	bulk snow redistribution scheme	`none`
	`ITD`	ITD snow redistribution scheme	
	`ITDrdg`	ITDrdg snow redistribution scheme	
	`none`	snow redistribution scheme off	
`snw_aging_table`	file	read 1D and 3D fields for dry metamorophsis lookup table	test
	snicar	read 3D fields for dry metamorophsis lookup table	
	test	internally generated dry metamorophsis lookup table for testing	
`snw_drdt0_fname`	string	snow aging file drdt0 fieldname	unknown
`snw_filename`	string	snow aging table data filename	unknown
`snw_kappa_fname`	string	snow aging file kappa fieldname	unknown
`snw_rhos_fname`	string	snow aging file rhos fieldname	unknown
`snw_T_fname`	string	snow aging file T fieldname	unknown
`snw_tau_fname`	string	snow aging file tau fieldname	unknown
`snw_Tgrd_fname`	string	snow aging file Tgrd fieldname	unknown
`use_smliq_pnd`	logical	use liquid in snow for ponds	`.false.`
`windmin`	real	minimum wind speed to compact snow in m/s	10.

forcing_nml

Table 23: **forcing_nml namelist options**

variable	options/format	description	default value
`atmbndy`	string	bulk transfer coefficients	`similarity`
	`similarity`	stability-based boundary layer	
	`constant`	constant-based boundary layer	
	`mixed`	stability-based boundary layer for wind stress, constant-based for sensible+latent heat fluxes	
`atmiter_conv`	real	convergence criteria for ustar	0.0
`atm_data_dir`	string	path or partial path to atmospheric forcing data directory	
`atm_data_format`	`bin`	read direct access binary atmo forcing file format	`bin`
	`nc`	read netcdf atmo forcing files	
`atm_data_type`	`box2001`	forcing data for [20] box problem	`default`
	`default`	constant values defined in the code	
	`hycom`	HYCOM atm forcing data in netCDF format	
	`JRA55`	JRA55 forcing data [61]	
	`JRA55do`	JRA55do forcing data [61]	
	`monthly`	monthly forcing data	
	`ncar`	NCAR bulk forcing data	
	`oned`	column forcing data	
`atm_data_version`	string	date of atm data forcing file creation	`_undef`
`bgc_data_dir`	string	path to oceanic forcing data directory	'unknown_bgc_data_dir'
`bgc_data_type`	`clim`	bgc climatological data	`default`
	`default`	constant values defined in the code	
	`hycom`	HYCOM ocean forcing data in netCDF format	
	`ncar`	POP ocean forcing data	
`calc_strair`	`.false.`	read wind stress and speed from files	`.true.`
	`.true.`	calculate wind stress and speed	
`calc_Tsfc`	logical	calculate surface temperature	`.true.`
`cpl_frazil`	`external`	frazil water/salt fluxes are handled outside of Icepack	`fresh_ice_correction`
	`fresh_ice_correction`	correct fresh-ice frazil water/salt fluxes for mushy physics	
	`internal`	send full frazil water/salt fluxes for mushy physics	
`default_season`	`summer`	forcing initial summer values	`winter`
	`winter`	forcing initial winter values	
`emissivity`	real	emissivity of snow and ice	0.985
`fbot_xfer_type`	`Cdn_ocn`	variable ocean heat transfer coefficient scheme	`constant`
	`constant`	constant ocean heat transfer coefficient	
`fe_data_type`	`clim`	ocean climatology forcing value for iron	`default`
	`default`	default forcing value for iron	
`formdrag`	logical	calculate form drag	`.false.`

continues on next page

Table 23 – continued from previous page

variable	options/format	description	default value
`fyear_init`	integer	first year of atmospheric forcing data	1900
`highfreq`	logical	high-frequency atmo coupling	`.false.`
`ice_data_conc`	`box2001`	ice distribution ramped from 0 to 1 west to east consistent with *box2001* test ([20])	`default`
	`c1`	initial ice concentation of 1.0	
	`default`	same as parabolic	
	`p5`	initial concentration of 0.5	
	`p8`	initial concentration of 0.8	
	`p9`	initial concentration of 0.9	
	`parabolic`	parabolic in ice thickness space with sum of aicen=1.0	
`ice_data_dist`	`box2001`	ice distribution ramped from 0 to 1 west to east consistent with *box2001* test ([20])	`default`
	`default`	uniform distribution, equivalent to uniform	
	`gauss`	gauss distbution of ice with a peak in the center of the domain	
	`uniform`	uniform distribution, equivalent to default	
`ice_data_type`	`block`	ice block covering about 25 percent of the area in center of domain	`default`
	`boxslotcyl`	slot cylinder ice mask associated with *boxslotcyl* test ([67])	
	`box2001`	box2001 ice mask associate with *box2001* test ([20])	
	`channel`	ice defined on entire grid in i-direction and 50% in j-direction in center of domain	
	`default`	same as latsst	
	`eastblock`	ice block covering about 25 percent of domain at the east edge of the domain	
	`latsst`	ice dependent on latitude and ocean temperature	
	`uniform`	ice defined at all grid points	
`ice_ref_salinity`	real	sea ice salinity for coupling fluxes (ppt)	4.0
`iceruf`	real	ice surface roughness at atmosphere interface in meters	0.0005
`l_mpond_fresh`	`.false.`	release pond water immediately to ocean	`.false.`
	`true`	retain (topo) pond water until ponds drain	
`natmiter`	integer	number of atmo boundary layer iterations	5
`nfreq`	integer	number of frequencies in ocean surface wave spectral forcing	25
`oceanmixed_file`	string	data file containing ocean forcing data	'unknown_oceanmixed_file'
`oceanmixed_ice`	logical	active ocean mixed layer calculation	`.false.`
`ocn_data_dir`	string	path to oceanic forcing data directory	'unknown_ocn_data_dir'
`ocn_data_format`	`bin`	read direct access binary ocean forcing files	`bin`
	`nc`	read netCDF ocean forcing files	
`ocn_data_type`	`clim`	ocean climatological data formulation	`default`
	`default`	constant values defined in the code	
	`hycom`	HYCOM ocean forcing data in netCDF format	

continues on next page

Table 23 – continued from previous page

variable	options/format	description	default value
	ncar	POP ocean forcing data	
precip_units	mks	liquid precipitation data units	mks
	mm_per_month		
	mm_per_sec	(same as MKS units)	
	m_per_sec		
restart_coszen	logical	read/write coszen in restart files	.false.
restore_ocn	logical	restore sst to data	.false.
restore_ice	logical	restore ice state along lateral boundaries	.false.
rotate_wind	logical	rotate wind from east/north to computation grid	.true.
saltflux_option	constant	computed using ice_ref_salinity	constant
	prognostic	computed using prognostic salinity	
tfrz_option	constant	constant ocean freezing temperature (Tocnfrz)	mushy
	linear_salt	linear function of salinity (ktherm=1)	
	minus1p8	constant ocean freezing temperature ($-1.8°C$)	
	mushy	matches mushy-layer thermo (ktherm=2)	
trestore	integer	sst restoring time scale (days)	90
ustar_min	real	minimum value of ocean friction velocity in m/s	0.0005
update_ocn_f	.false.	do not include frazil water/salt fluxes in ocn fluxes	.false.
	true	include frazil water/salt fluxes in ocn fluxes	
wave_spec_file	string	data file containing wave spectrum forcing data	
wave_spec_type	constant	wave data file is provided, constant wave spectrum, for testing	none
	none	no wave data provided, no wave-ice interactions	
	profile	no wave data file is provided, use fixed dummy wave spectrum, for testing	
	random	wave data file is provided, wave spectrum generated using random number	
ycycle	integer	number of years in forcing data cycle	1

zbgc_nml

Table 24: **zbgc_nml namelist options**

variable	options/format	description	default value
algaltype_diatoms	real	mobility type between stationary and mobile algal diatoms	0.0
algaltype_phaeo	real	mobility type between stationary and mobile algal phaeocystis	0.5
algaltype_sp	real	mobility type between stationary and mobile small plankton	0.5

continues on next page

Table 24 – continued from previous page

variable	options/format	description	default value
algal_vel	real	[32]	1.11e-8
alpha2max_low_diatoms	real	light limitation diatoms 1/(W/m^2)	0.8
alpha2max_low_phaeo	real	light limitation phaeocystis 1/(W/m^2)	0.67
alpha2max_low_sp	real	light limitation small plankton 1/(W/m^2)	0.67
ammoniumtype	real	mobility type between stationary and mobile ammonium	1.0
beta2max_diatoms	real	light inhibition diatoms 1/(W/m^2)	0.18
beta2max_phaeo	real	light inhibition phaeocystis 1/(W/m^2)	0.01
beta2max_sp	real	light inhibition small plankton 1/(W/m^2)	0.0025
bgc_flux_type	constant	constant ice–ocean flux velocity	Jin2006
	Jin2006	ice–ocean flux velocity of [26]	
chlabs_diatoms	real	chl absorbtion diatoms 1/m/(mg/m^3)	0.03
chlabs_phaeo	real	chl absorbtion phaeocystis 1/m/(mg/m^3)	0.05
chlabs_sp	real	chl absorbtion small plankton 1/m/(mg/m^3)	0.01
dEdd_algae	logical		.false.
dmspdtype	real	mobility type between stationary and mobile dmspd	-1.0
dmspptype	real	mobility type between stationary and mobile dmspp	0.5
doctype_l	real	mobility type between stationary and mobile doc lipids	0.5
doctype_s	real	mobility type between stationary and mobile doc saccharids	0.5
dontype_protein	real	mobility type between stationary and mobile don proteins	0.5
dustFe_sol	real	solubility fraction	0.005
fedtype_1	real	mobility type between stationary and mobile fed lipids	0.5
feptype_1	real	mobility type between stationary and mobile fep lipids	0.5
frazil_scav	real	increase in initial bio bracer from ocean scavenging	1.0
fr_dFe	real	fraction of remineralized nitrogen in units of algal iron	0.3
fr_graze_diatoms	real	fraction grazed diatoms	0.01
fr_graze_e	real	fraction of assimilation excreted	0.5
fr_graze_phaeo	real	fraction grazed phaeocystis	0.1
fr_graze_s	real	fraction of grazing spilled or slopped	0.5
fr_graze_sp	real	fraction grazed small plankton	0.1
fr_mort2min	real	fractionation of mortality to Am	0.5
fr_resp	real	frac of algal growth lost due to respiration	0.05
fr_resp_s	real	DMSPd fraction of respiration loss as DMSPd	0.75
fsal	real	salinity limitation ppt	1.0
F_abs_chl_diatoms	real	scales absorbed radiation for dEdd chl diatoms	2.0
F_abs_chl_phaeo	real	scales absorbed radiation for dEdd chl phaeocystis	5.0

continues on next page

Table 24 – continued from previous page

variable	options/format	description	default value
F_abs_chl_sp	real	scales absorbed radiation for dEdd small plankton	4.0
f_doc_l	real	fraction of mortality to DOC lipids	0.4
f_doc_s	real	fraction of mortality to DOC saccharides	0.4
f_don_Am_protein	real	fraction of remineralized DON to ammonium	0.25
f_don_protein	real	fraction of spilled grazing to proteins	0.6
f_exude_l	real	fraction of exudation to DOC lipids	1.0
f_exude_s	real	fraction of exudation to DOC saccharids	1.0
grid_o	real	z biology for bottom flux	5.0
grid_o_t	real	z biology for top flux	5.0
grid_oS	real	zsalinity DEPRECATED	
grow_Tdep_diatoms	real	temperature dependence growth diatoms per degC	0.06
grow_Tdep_phaeo	real	temperature dependence growth phaeocystis per degC	0.06
grow_Tdep_sp	real	temperature dependence growth small plankton per degC	0.06
humtype	real	mobility type between stationary and mobile hum	1.0
initbio_frac	real	fraction of ocean trcr concentration in bio tracers	1.0
K_Am_diatoms	real	ammonium half saturation diatoms mmol/m^3	0.3
K_Am_phaeo	real	ammonium half saturation phaeocystis mmol/m^3	0.3
K_Am_sp	real	ammonium half saturation small plankton mmol/m^3	0.3
k_bac_l	real	Bacterial degredation of DOC lipids per day	0.03
k_bac_s	real	Bacterial degredation of DOC saccharids per day	0.03
k_exude_diatoms	real	algal exudation diatoms per day	0.0
k_exude_phaeo	real	algal exudation phaeocystis per day	0.0
k_exude_sp	real	algal exudation small plankton per day	0.0
K_Fe_diatoms	real	iron half saturation diatoms nM	1.0
K_Fe_phaeo	real	iron half saturation phaeocystis nM	0.1
K_Fe_sp	real	iron half saturation small plankton nM	0.2
k_nitrif	real	nitrification rate per day	0.0
K_Nit_diatoms	real	nitrate half saturation diatoms mmol/m^3	1.0
K_Nit_phaeo	real	nitrate half saturation phaeocystis mmol/m^3	1.0
K_Nit_sp	real	nitrate half saturation small plankton mmol/m^3	1.0
K_Sil_diatoms	real	silicate half saturation diatoms mmol/m^3	4.0
K_Sil_phaeo	real	silicate half saturation phaeocystis mmol/m^3	0.0
K_Sil_sp	real	silicate half saturation small plankton mmol/m^3	0.0
kn_bac_protein	real	bacterial degradation of DON per day	0.03
l_sk	real	characteristic diffusive scale in m	7.0

continues on next page

Table 24 – continued from previous page

variable	options/format	description	default value
l_skS	real	zsalinity DEPRECATED	
max_dfe_doc1	real	max ratio of dFe to saccharides in the ice in nm Fe / muM C	0.2
max_loss	real	restrict uptake to percent of remaining value	0.9
modal_aero	logical	modal aerosols	.false.
mort_pre_diatoms	real	mortality diatoms	0.007
mort_pre_phaeo	real	mortality phaeocystis	0.007
mort_pre_sp	real	mortality small plankton	0.007
mort_Tdep_diatoms	real	temperature dependence of mortality diatoms per degC	0.03
mort_Tdep_phaeo	real	temperature dependence of mortality phaeocystis per degC	0.03
mort_Tdep_sp	real	temperature dependence of mortality small plankton per degC	0.03
mu_max_diatoms	real	maximum growth rate diatoms per day	1.2
mu_max_phaeo	real	maximum growth rate phaeocystis per day	0.851
mu_max_sp	real	maximum growth rate small plankton per day	0.851
nitratetype	real	mobility type between stationary and mobile nitrate	-1.0
op_dep_min	real	light attenuates for optical depths exceeding min	0.1
phi_snow	real	snow porosity for brine height tracer	0.5
ratio_chl2N_diatoms	real	algal chl to N in mg/mmol diatoms	2.1
ratio_chl2N_phaeo	real	algal chl to N in mg/mmol phaeocystis	0.84
ratio_chl2N_sp	real	algal chl to N in mg/mmol small plankton	1.1
ratio_C2N_diatoms	real	algal C to N in mol/mol diatoms	7.0
ratio_C2N_phaeo	real	algal C to N in mol/mol phaeocystis	7.0
ratio_C2N_proteins	real	algal C to N in mol/mol proteins	7.0
ratio_C2N_sp	real	algal C to N in mol/mol small plankton	7.0
ratio_Fe2C_diatoms	real	algal Fe to C in umol/mol diatoms	0.0033
ratio_Fe2C_phaeo	real	algal Fe to C in umol/mol phaeocystis	1.0
ratio_Fe2C_sp	real	algal Fe to C in umol/mol small plankton	0.0033
ratio_Fe2N_diatoms	real	algal Fe to N in umol/mol diatoms	0.23
ratio_Fe2N_phaeo	real	algal Fe to N in umol/mol phaeocystis	0.7
ratio_Fe2N_sp	real	algal Fe to N in umol/mol small plankton	0.23
ratio_Fe2DOC_s	real	Fe to C of DON saccharids nmol/umol	1.0
ratio_Fe2DOC_l	real	Fe to C of DOC lipids nmol/umol	0.033
ratio_Fe2DON	real	Fe to C of DON nmol/umol	0.023
ratio_Si2N_diatoms	real	algal Si to N in mol/mol diatoms	1.8
ratio_Si2N_phaeo	real	algal Si to N in mol/mol phaeocystis	0.0
ratio_Si2N_sp	real	algal Si to N in mol/mol small plankton	0.0
ratio_S2N_diatoms	real	algal S to N in mol/mol diatoms	0.03
ratio_S2N_phaeo	real	algal S to N in mol/mol phaeocystis	0.03
ratio_S2N_sp	real	algal S to N in mol/mol small plankton	0.03
restart_bgc	logical	restart tracer values from file	.false.
restart_hbrine	logical		.false.
restart_zsal	logical	zsalinity DEPRECATED	.false.
restore_bgc	logical	restore bgc to data	.false.

continues on next page

Table 24 – continued from previous page

variable	options/format	description	default value
R_dFe2dust	real	g/g [57]	0.035
scale_bgc	logical		.false.
silicatetype	real	mobility type between stationary and mobile silicate	-1.0
skl_bgc	logical	biogeochemistry	.false.
solve_zbgc	logical		.false.
solve_zsal	logical	zsalinity DEPRECATED, update salinity tracer profile	.false.
tau_max	real	long time mobile to stationary exchanges	1.73e-5
tau_min	real	rapid module to stationary exchanges	5200.
tr_bgc_Am	logical	ammonium tracer	.false.
tr_bgc_C	logical	algal carbon tracer	.false.
tr_bgc_chl	logical	algal chlorophyll tracer	.false.
tr_bgc_DMS	logical	DMS tracer	.false.
tr_bgc_DON	logical	DON tracer	.false.
tr_bgc_Fe	logical	iron tracer	.false.
tr_bgc_hum	logical		.false.
tr_bgc_Nit	logical		.false.
tr_bgc_PON	logical	PON tracer	.false.
tr_bgc_Sil	logical	silicate tracer	.false.
tr_brine	logical	brine height tracer	.false.
tr_zaero	logical	vertical aerosol tracers	.false.
t_iron_conv	real	desorption loss pFe to dFe in days	3065.
t_sk_conv	real	Stefels conversion time in days	3.0
t_sk_ox	real	DMS oxidation time in days	10.0
T_max	real	maximum temperature degC	0.0
y_sk_DMS	real	fraction conversion given high yield	0.5
zaerotype_bc1	real	mobility type between stationary and mobile zaero bc1	1.0
zaerotype_bc2	real	mobility type between stationary and mobile zaero bc2	1.0
zaerotype_dust1	real	mobility type between stationary and mobile zaero dust1	1.0
zaerotype_dust2	real	mobility type between stationary and mobile zaero dust2	1.0
zaerotype_dust3	real	mobility type between stationary and mobile zaero dust3	1.0
zaerotype_dust4	real	mobility type between stationary and mobile zaero dust4	1.0
z_tracers	logical		.false.

icefields_nml

There are several icefield namelist groups to control model history output. See the source code for a full list of supported output fields.

- `icefields_nml` is in **cicecore/cicedyn/analysis/ice_history_shared.F90**
- `icefields_bgc_nml` is in **cicecore/cicedyn/analysis/ice_history_bgc.F90**
- `icefields_drag_nml` is in **cicecore/cicedyn/analysis/ice_history_drag.F90**
- `icefields_fsd_nml` is in **cicecore/cicedyn/analysis/ice_history_fsd.F90**
- `icefields_mechred_nml` is in **cicecore/cicedyn/analysis/ice_history_mechred.F90**
- `icefields_pond_nml` is in **cicecore/cicedyn/analysis/ice_history_pond.F90**
- `icefields_snow_nml` is in **cicecore/cicedyn/analysis/ice_history_snow.F90**

Table 25: **icefields_nml namelist options**

variable	options/format	description	default value
`f_<var>`	`d`	write field var every `histfreq_n` days	
	`h`	write field var every `histfreq_n` hours	
	`m`	write field var every `histfreq_n` months	
	`x`	do not write var to history	
	`y`	write field var every `histfreq_n` years	
	`1`	write field var every time step	
	`md`	*e.g.,* write both monthly and daily files	
`f_<var>_ai`	`d`	write field cell average var every `histfreq_n` days	
	`h`	write field cell average var every `histfreq_n` hours	
	`m`	write field cell average var every `histfreq_n` months	
	`x`	do not write cell average var to history	
	`y`	write field cell average var every `histfreq_n` years	
	`1`	write field cell average var every time step	
	`md`	*e.g.,* write both monthly and daily files	

3.5 Troubleshooting

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

3.5.1 Directory Structure

In November, 2022, the cicedynB directory was renamed to cicedyn.

3.5.2 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.

3.5.3 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 NetCDF 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.

3.5.4 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.

3.5.5 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 **(ice_diagnostics.F90)**
: A wrapper for *print_state* that is easily called from numerous points during the timestepping loop.

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

debug_forcing **= true (ice_in)**
: Print numerous diagnostic quantities associated with input forcing.

debug_blocks **= true (ice_in)**
: Print diagnostics during block decomposition and distribution.

debug_model **= true (ice_in)**
: Print extended diagnostics for the first point associated with *print_points*.

debug_model_i **= integer (ice_in)**
: Defines the local i index for the point to be diagnosed with *debug_model*.

debug_model_j **= integer (ice_in)**
: Defines the local j index for the point to be diagnosed with *debug_model*.

debug_model_iblk **= integer (ice_in)**
: Defines the local iblk value for the point to be diagnosed with *debug_model*.

debug_model_task **= integer (ice_in)**
: Defines the local task value for the point to be diagnosed with *debug_model*.

debug_model_step **= true (ice_in)**
: Timestep to starting printing diagnostics associated with *debug_model*.

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, defined by *lonpnt* and *latpnt* in the namelist file. 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 to call *print_state*. This option can be fairly slow, due to gathering data from processors.

conserv_check **= true (ice_in)**
: Diagnoses conservation in various algorithms.

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.

3.5.6 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.

3.5.7 Interpretation of albedos

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

3.5.8 VP dynamics results

The VP dynamics solver (*kdyn=3*) requires a global sum. This global sum is computed by default via an efficient implementation that is not bit-for-bit for different decompositions or pe counts. Bit-for-bit identical results can be recovered for the VP dynamics solver by setting the namelist *bfbflag* = *reprosum* or using the *-s reprosum* option when setting up a case.

3.5.9 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

FOUR

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

4.1.2 Coding Standard

Overall, CICE code should be implemented as follows,

- Adhere to the current coding and naming conventions
- Write readable code. Use meaningful variable names; indent 2 or 3 spaces for loops and conditionals; vertically align similar elements where it makes sense, and provide concise comments throughout the code.
- Declare common parameters in a shared module. Do not hardwire the same parameter in the code in multiple places.
- Maintain bit-for-bit output for the default configuration (to the extent possible). Use namelist options to add new features.
- Maintain global conservation of heat, water, salt

- Use of C preprocessor (CPP) directives should be minimized and only used for build dependent modifications such as use of netcdf (or other "optional" libraries) or for various Fortran features that may not be supported by some compilers. Use namelist to support run-time code options. CPPs should be all caps.
- All modules should have the following set at the top

  ```
  implicit none
  private
  ```

  Any public module interfaces or data should be explicitly specified
- All subroutines and functions should define the `subname` character parameter statement to match the interface name like

  ```
  character(len=*),parameter :: subname='(advance_timestep)'
  ```

- Public Icepack interfaces should be accessed thru the `icepack_intfc` module like

  ```
  use icepack_intfc, only: icepack_init_parameters
  ```

- Icepack does not write to output or abort, it provides methods to access those features. After each call to Icepack, **icepack_warnings_flush** should be called to flush Icepack output to the CICE log file and **icepack_warnings_aborted** should be check to abort on an Icepack error as follows,

  ```
  call icepack_physics()
  call icepack_warnings_flush(nu_diag)
  if (icepack_warnings_aborted()) call abort_ice(error_message=subname, file=__FILE__,
  ↪ line=__LINE__)
  ```

- Use `ice_check_nc` or `ice_pio_check` after netcdf or pio calls to check for return errors.
- Use subroutine `abort_ice` to abort the model run. Do not use stop or MPI_ABORT. Use optional arguments (file=__FILE__, line=__LINE__) in calls to `abort_ice` to improve debugging
- Write output to stdout from the master task only unless the output is associated with an abort call. Write to unit `nu_diag` following the current standard. Do not use units 5 or 6. Do not use the print statement.
- Use of new Fortran features or external libraries need to be balanced against usability and the desire to compile on as many machines and compilers as possible. Developers are encouraged to contact the Consortium as early as possible to discuss requirements and implementation in this case.

4.2 Dynamics

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

cicecore/cicedyn/analysis contains higher level history and diagnostic routines.

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

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

cicecore/cicedyn/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 Dynamical Solvers

The dynamics solvers are found in **cicecore/cicedyn/dynamics/**. A couple of different solvers are available including EVP, EAP and VP. The dynamics solver is specified in namelist with the `kdyn` variable. `kdyn=1` is evp, `kdyn=2` is eap, `kdyn=3` is VP.

Two alternative implementations of EVP are included. The first alternative is the Revised EVP, triggered when the `revised_evp` is set to true. The second alternative is the 1d EVP solver triggered when the `evp_algorithm` is set to `shared_mem_1d` as oppose to the default setting of `evp_standard_2d`. The solutions with `evp_algorithm` set to `standard_2d` or `shared_mem_1d` will not be bit-for-bit identical when compared to each other. The reason for this is floating point round off errors that occur unless strict compiler flags are used. `evp_algorithm=shared_mem_1d` is primarily built for OpenMP. If MPI domain splitting is used then the solver will only run on the master processor. `evp_algorithm=shared_mem_1d` is not supported with the tripole grid.

4.2.2 Transport

The transport (advection) methods are found in **cicecore/cicedyn/dynamics/**. Two methods are supported, upwind and remap. These are set in namelist via the `advection` variable. Transport can be disabled with the `ktransport` namelist variable.

4.3 Infrastructure

4.3.1 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.

4.3.2 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.

4.3.3 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 *Tables 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

4.3.4 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.

The time manager was updated in early 2021. Additional information about the time manager can be found here, *Time Manager and Initialization*.

4.3.5 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.

cicedyn/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.

cicedyn/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.

4.3.6 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.

cicedyn/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.

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

cicedyn/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.

There is additional IO information in *Model Input and Output*.

4.4 Driver and Coupling

The driver and coupling layer is found in **cicecore/drivers/**. The standalone driver is found under **cicecore/drivers/standalone/cice/** and other high level coupling layers are found in other directories. CICE is designed to build with only one of these drivers at a time, depending how the model is run and coupled. Within the **cicecore/drivers/standalone/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.

The files provide the top level sequencing for calling the standalone CICE model.

4.4.1 Adding a New Driver

The drivers directory contains two levels of subdirectories. The first layer indicates the coupling infrastructure or strategy and the second later indicates the application or coupler the driver is written for. At the present time, the directory structures is:

```
drivers/direct/hadgem3
drivers/mct/cesm1
drivers/nuopc/cmeps
drivers/standalone/cice
```

The standalone driver is **drivers/standalone/cice**, and this is the driver used when running with the CICE scripts in standalone mode. New drivers can be added as needed when coupling to new infrastructure or in new applications. We encourage the community to use the drivers directory to facilitate reuse with the understanding that the driver code could also reside in the application. Users should follow the naming strategy as best as possible. Drivers should be added under the appropriate subdirectory indicative of the coupling infrastructure. New subdirectories (such as oasis or esmf) can be added in the future as needed. The community will have to decide when it's appropriate to share drivers between different applications, when to update drivers, and when to create new drivers. There are a number of trade-offs to consider including backwards compatibility with earlier versions of applications, code reuse, and independence. As a general rule, driver directories should not be deleted and names should not be reused to avoid confusion with prior versions that were fundamentally different. The number of drivers will likely increase over time as new infrastructure and applications are added and as versions evolve in time.

The current drivers subdirectories are mct, nuopc, standalone, and direct. The standalone subdirectory contains drivers to run the model in standalone mode as a standalone program. The direct subdirectory contains coupling interfaces that supporting calling the ice model directory from other models as subroutines. The subdirectory mct contains subdirectories for applications/couplers that provide coupling via mct interfaces. And the subdirectory nuopc contains subdirectories for applications/couplers that provide coupling via nuopc interfaces.

The varied **cicecore/drivers/** directories are generally implemented similar to the standalone cice case with versions of **CICE_InitMod.F90**, **CICE_RunMod.F90**, and **CICE_FinalMod.F90** files in addition to files consistent with the coupling layer.

As features are added to the CICE model over time that require changes in the calling sequence, it's possible that all drivers will need to be updated. These kinds of changes are impactful and not taken lightly. It will be up to the community as a whole to work together to maintain the various drivers in these situations.

4.4.2 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 count_tracers        ! count tracers
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_zbgc            ! vertical biogeochemistry initialization
call init_calendar        ! initialize some calendar stuff
call init_hist (dt)       ! initialize output history file
call init_dyn (dt_dyn)    ! define dynamics parameters, variables
if (kdyn == 2) then
   call init_eap          ! define eap dynamics parameters, variables
else if (kdyn == 3) then
   call init_vp           ! define vp 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
if (tr_fsd) call icepack_init_fsd_bounds  ! floe size distribution
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
call init_history_dyn     ! initialize dynamic history variables
call calc_timesteps       ! update timestep counter if not using npt_unit="1"
call init_shortwave       ! initialize radiative transfer
call advance_timestep     ! advance the time step
call init_forcing_atmo    ! initialize atmospheric forcing (standalone)
if (tr_fsd .and. wave_spec) call get_wave_spec ! wave spectrum in ice
call get_forcing*         ! read forcing data (standalone)
if (tr_snow) call icepack_init_snow ! advanced snow physics
```

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)
```

(continues on next page)

(continued from previous page)

```
   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)

if (tr_fsd .and. wave_spec) call step_dyn_wave(dt)
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

if (tr_snow) then         ! advanced snow physics
   do iblk = 1, nblocks
      call step_snow (dt, iblk)
   enddo
   call update_state (dt) ! clean up
endif

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

! write data
! update forcing
```

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

4.5 Standalone Forcing

Users are strongly encouraged to run CICE in a coupled system (see *Coupling With Other Climate Model Components*) to improve quality of science. The standalone mode is best used for technical testing and only preliminary science testing. Several different input forcing datasets have been implemented over the history of CICE. Some have become obsolete, others have been supplanted by newer forcing data options, and others have been implemented by outside users and are not testable by the Consortium. The forcing code has generally not been maintained by the Consortium and only a subset of the code is tested by the Consortium.

The forcing implementation can be found in the file **cicecore/cicedyn/general/ice_forcing.F90**. As noted above, only a subset of the forcing modes are tested and supported. In many ways, the implemetation is fairly primitive, in part due to historical reasons and in part because standalone runs are discouraged for evaluating complex science. In general, most implementations use aspects of the following approach,

- Input files are organized by year. The underlying implementation provides for some flexibility and extensibility in filenames. For instance, JRA55 and JRA55do filenames can have syntax like [JRA55,JRA55do][_$grid]_03hr_forcing_$year.nc or [JRA55,JRA55do]_03hr_forcing[_$grid]_$year.nc, where [_$grid] is optional and may be present at one of two locations within the filename. This implementation exists to support the current naming conventions within the gx1, gx3, and tx1 JRA55 and JRA55do CICE_data directory structure automatically. See **JRA55_files** in **ice_forcing.F90** for more details.- Namelist inputs `fyear` and `ycycle` specify the forcing year dataset.

- The forcing year is computed on the fly and is assumed to be cyclical over the forcing dataset length defined by `ycycle`.
- The namelist `atm_data_dir` specifies the full or partial path for the atmosphere input data files, and the namelist `atm_data_type` defines the atmospheric forcing mode (see `forcing_nml` in *Tables of Namelist Options*). Many of the forcing options are generated internally. For atmospheric forcing read from files, the directory structure and filenames depend on the grid and `atm_data_type`. Many details can be gleaned from the CICE_data directory and filenames as well as from the implementation in **ice_forcing.F90**. The primary `atm_data_type` forcing for gx1, gx3, and tx1 test grids are `JRA55` and `JRA55do`. For those configurations, the `atm_data_dir` should be set to ${CICE_data_root}/forcing/${grid}/[JRA55,JRA55do] and the filenames should be of the form [JRA55,JRA55do]_${grid}_03hr_forcing${atm_data_version}_yyyy.nc where yyyy is the forcing year. Those files should be placed under `atm_data_dir/8XDAILY`. `atm_data_version` is a string defined in `forcing_nml` namelist that supports versioning of the forcing data. `atm_data_version` could be any string including the null string. It typically will be something like _yyyymmdd to indicate the date the forcing data was generated.
- The namelist `ocn_data_dir` specifies the directory of the ocean input data files and the namelist `ocn_data_type` defines the ocean forcing mode.
- The filenames follow a particular naming convention that is defined in the source code (ie. subroutine **JRA55_files**). The forcing year is typically found just before the **.nc** part of the filename and there are tools (subroutine **file_year**) to update the filename based on the model year and appropriate forcing year.
- The input data time axis is generally NOT read by the forcing subroutine. The forcing frequency is hardwired into the model and the file record number is computed based on the forcing frequency and model time. Mixing leap year input data and noleap model calendars (and vice versa) is not handled particularly gracefully. The CICE model does not read or check against the input data time axis.
- Data is read on the model grid, no spatial interpolation exists.
- Data is often time interpolated linearly between two input timestamps to the model time each model timestep.

In general, the following variables need to be defined by the forcing module,

From Atmosphere:

- zlvl = atmosphere level height (m)
- uatm = model grid i-direction wind velocity component (m/s)
- vatm = model grid j-direction wind velocity component (m/s)
- strax = model grid i-direction wind stress (N/m^2)
- stray = model grid j-direction wind stress (N/m^2)
- potT = air potential temperature (K)
- Tair = air temperature (K)
- Qa = specific humidity (kg/kg)
- rhoa = air density (kg/m^3)
- flw = incoming longwave radiation (W/m^2)
- fsw = incoming shortwave radiation (W/m^2)
- swvdr = sw down, visible, direct (W/m^2)
- swvdf = sw down, visible, diffuse (W/m^2)
- swidr = sw down, near IR, direct (W/m^2)

- swidf = sw down, near IR, diffuse (W/m^2)
- frain = rainfall rate (kg/m^2 s)
- fsnow = snowfall rate (kg/m^2 s)

From Ocean:

- uocn = ocean current, x-direction (m/s)
- vocn = ocean current, y-direction (m/s)
- ss_tltx = sea surface slope, x-direction (m/m)
- ss_tlty = sea surface slope, y-direction (m/m)
- hwater = water depth for basal stress calc (landfast ice)
- sss = sea surface salinity (ppt)
- sst = sea surface temperature (C)
- frzmlt = freezing/melting potential (W/m^2)
- frzmlt_init= frzmlt used in current time step (W/m^2)
- Tf = freezing temperature (C)
- qdp = deep ocean heat flux (W/m^2), negative upward
- hmix = mixed layer depth (m)
- daice_da= data assimilation concentration increment rate (concentration s-1)(only used in hadgem drivers)

All variables have reasonable but static defaults and these will be used in `default` mode.

To advance the forcing, the subroutines **get_forcing_atmo** and **get_forcing_ocn** are called each timestep from the step loop. That subroutine computes the forcing year (`fyear`), calls the appropriate forcing data method, and then calls **prepare_forcing** which converts the input data fields to model forcing fields.

4.5.1 JRA55 and JRA55do Atmosphere Forcing

The current default atmosphere forcing for gx3, gx1, and tx1 standalone grids for Consortium testing is the JRA55 forcing dataset [61]. The Consortium has released 5 years of forcing data, 2005-2009 for gx3, gx1, and tx1 grids. Each year is a separate file and the dataset is on a gregorian time axis which includes leap days.

The forcing is read and interpolated in subroutine **JRA55_data**. In particular, air temperature (`airtmp`), east and north wind speed (`wndewd` and `wndnwd`), specific humidity (`spchmd`), incoming short and longwave radiation (`glbrad` and `dswsfc`), and precipitation (`ttlpcp`) are read from the input files. The JRA55 reanalysis is run with updated initial conditions every 6 hours and output is written every 3 hours. The four state fields (air temperature, winds, specific humidity) are instantaneous data, while the three flux fields (radition, precipitation) are 3 hour averages. In the JRA55 forcing files provided by the Consortium, the time defined for 3 hour average fields is shifted 3 hours to the start time of the 3 hour interval. **NOTE that this is different from the implementation on the original JRA55 files and also different from how models normally define time on an accumulated/averaged field**. This is all shown schematically in Figure *Schematic of JRA55 CICE forcing file generation.*.

The state fields are linearly time interpolated between input timestamps while the flux fields are read and held constant during each 3 hour model period. The forcing frequency is hardwired to 3 hours in the implementation, and the record number is computed based on the time of the current model year. Time interpolation coefficients are computed in the **JRA55_data** subroutine.

The forcing data is converted to model inputs in the subroutine **prepare_forcing** called in **get_forcing_atmo**. To clarify, the JRA55 input data includes

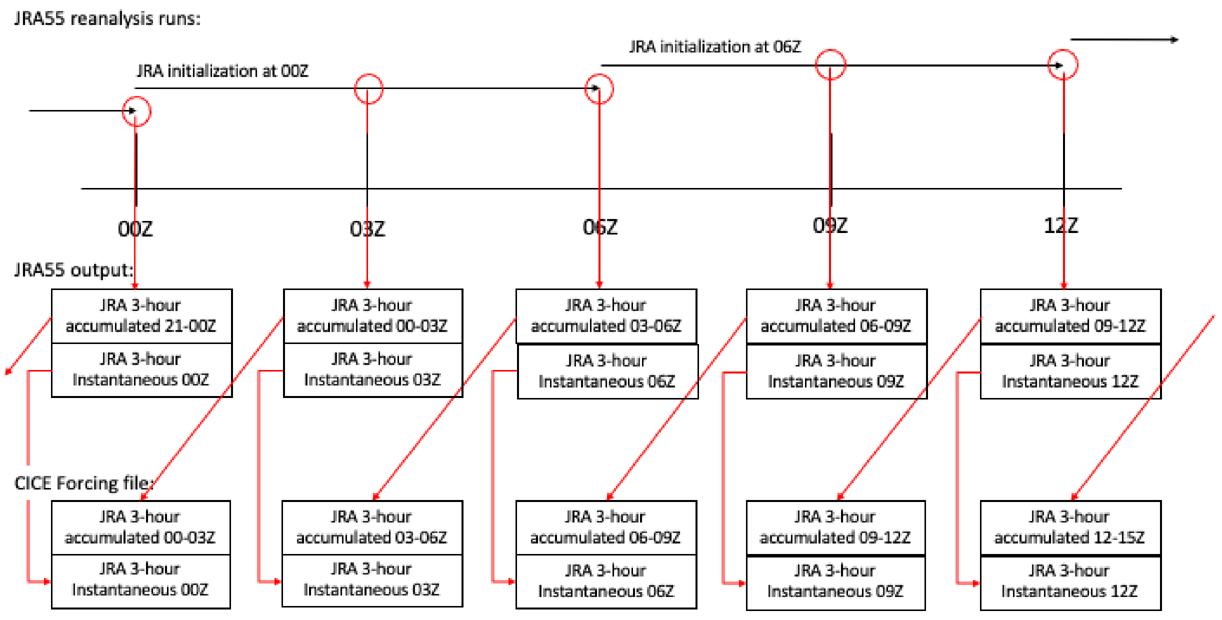

JRA55 reanalysis runs:
JRA initialization at 00Z
JRA initialization at 06Z
00Z
03Z
06Z
09Z
12Z
JRA55 output:
JRA 3-hour accumulated 21-00Z
JRA 3-hour Instantaneous 00Z
JRA 3-hour accumulated 00-03Z
JRA 3-hour Instantaneous 03Z
JRA 3-hour accumulated 03-06Z
JRA 3-hour Instantaneous 06Z
JRA 3-hour accumulated 06-09Z
JRA 3-hour Instantaneous 09Z
JRA 3-hour accumulated 09-12Z
JRA 3-hour Instantaneous 12Z
CICE Forcing file:
JRA 3-hour accumulated 00-03Z
JRA 3-hour Instantaneous 00Z
JRA 3-hour accumulated 03-06Z
JRA 3-hour Instantaneous 03Z
JRA 3-hour accumulated 06-09Z
JRA 3-hour Instantaneous 06Z
JRA 3-hour accumulated 09-12Z
JRA 3-hour Instantaneous 09Z
JRA 3-hour accumulated 12-15Z
JRA 3-hour Instantaneous 12Z

Fig. 1: Schematic of JRA55 CICE forcing file generation.

- uatm = T-cell centered, model grid i-direction wind velocity component (m/s)
- vatm = T-cell-centered, model grid j-direction wind velocity component (m/s)
- Tair = air temperature (K)
- Qa = specific humidity (kg/kg)
- flw = incoming longwave radiation (W/m^2)
- fsw = incoming shortwave radiation (W/m^2)
- fsnow = snowfall rate (kg/m^2 s)

and model forcing inputs are derived from those fields and the defaults.

Because the input files are on the gregorian time axis, the model can run with the regular 365 day (noleap) calendar, but in that case, the Feb 29 input data will be used on March 1, and all data after March 1 will be shifted one day. December 31 in leap years will be skipped when running with a CICE calendar with no leap days.

JRA55do forcing is also provided by the Consortium in the same format and scheme. The JRA55do dataset is more focused on forcing for ocean and ice models, but provides a very similar climate as the JRA55 forcing. To switch to JRA55do, set the namelist `atm_data_type` to `JRA55do` and populate the input data directory with the JRA55do dataset provided by the Consortium.

4.5.2 NCAR Atmosphere Forcing

The NCAR atmospheric forcing was used in earlier standalone runs on the gx3 grid, and the Consortium continues to do some limited testing with this forcing dataset. Monthly average data for fsw, cldf, fsnow are read. 6-hourly data for Tair, uatm, vatm, rhoa, and Qa are also read. Users are encouraged to switch to the JRA55 (see *JRA55 and JRA55do Atmosphere Forcing*) dataset. This atmosphere forcing dataset may be deprecated in the future.

4.5.3 Default Atmosphere Forcing

The default atmosphere forcing option sets the atmosphere forcing internally. No files are read. Values for forcing fields are defined at initialization in subroutine **init_coupler_flux** and held constant thereafter. Different conditions can be specified thru the `default_season` namelist variable.

4.5.4 Box2001 Atmosphere Forcing

The box2001 forcing dataset in generated internally. No files are read. The dataset is used to test an idealized box case as defined in [20].

4.5.5 Other Atmosphere Forcing

There are a few other atmospheric forcing modes, as defined by `atm_data_type`, but they are not tested by the Consortium on a regular basis.

4.5.6 Default Ocean Forcing

The `default` ocean setting is the standard setting used in standalone CICE runs. In this mode, the sea surface salinity is set to 34 ppt and the sea surface temperature is set to the freezing temperature at all grid points and held constant unless the mixed layer parameterization is turned on, in which case the SST evolves. Other ocean coupling fields are set to zero. No files are read.

4.5.7 Other Ocean Forcing

There are a few other ocean forcing modes, as defined by `ocn_data_type`, but they are not tested by the Consortium on a regular basis.

4.6 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_flush** is called after every call to an Icepack method. It does not directly "use" Icepack data, accessing Icepack data only thru interfaces.

4.7 Scripts

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

4.7.1 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
- **timeseries.csh** generates PNG timeseries plots from output files, using GNUPLOT
- **timeseries.py** generates PNG timeseries plots from output files, using Python
- **tests/** scripts for configuring and running basic tests

4.7.2 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.

4.7.3 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. The name of the option (i.e. diag1, debug, bgcISPOL) should not have any special characters in the name as this can impact scripts usage.

4.7.4 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.

The **cice.build** script has some additional features including the ability to pass a Makefile target. This is documented in *More about cice.build*. In addition, there is a hidden feature in the **cice.build** script that allows for reuse of executables. This is used by the test suites to significantly reduce cost of building the model. It is invoked with the `--exe` argument to **cice.build** and should not be invoked by users interactively.

4.7.5 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*.

4.7.6 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.

4.7.7 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 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.

4.7.8 QC Process Validation

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

Two scripts have been created to automatically validate the QC 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 QC 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_test`, and `qc_fail`. `qc_base` is the base test case with the default QC namelist. `qc_bfb` is identical to `qc_base`. `qc_test` 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 QC 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>

# 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
```

4.8 Tools

4.8.1 CICE4 restart conversion

There is a Fortran program in **configuration/tools/cice4_restart_conversion** that will help convert a CICE4 restart file into a CICE5 restart file. There is a bit of documentation contained in that source code about how to build, use, and run the tool. A few prognostic variables were changed from CICE4 to CICE5 which fundamentally altered the fields saved to the restart file. See **configuration/tools/cice4_restart_conversion/convert_restarts.f90** for additional information.

4.8.2 JRA55 forcing datasets

This section describes how to generate JRA55 forcing data for the CICE model. Raw JRA55 or JRA55do files have to be interpolated and processed into input files specifically for the CICE model. A tool exists in **configuration/tools/jra55_datasets** to support that process. The raw JRA55 or JRA55do data is obtained from the NCAR/UCAR Research Data Archive and the conversion tools are written in python.

Requirements

Python3 is required, and the following python packages are required with the tested version number in parenthesis. These versions are not necessarily the only versions that work, they just indicate what versions were used when the script was recently run.

- python3 (python3.7.9)
- numpy (1.18.5)
- netCDF4 (1.5.5)
- ESMPy (8.0.0)
- xesmf (0.3.0)

NCO is required for aggregating the output files into yearly files.

- netcdf (4.7.4)
- nco (4.9.5)

Raw JRA55 forcing data

The raw JRA55 forcing data is obtained from the UCAR/NCAR Research Data Archive, https://rda.ucar.edu/. You must first register (free) and then sign in. The "JRA-55 Reanalysis Daily 3-Hourly and 6-Hourly Data" is ds628.0 and can be found here, https://rda.ucar.edu/datasets/ds628.0.

The "Data access" tabs will provide a list of product categories. The JRA55 data of interest are located in 2 separate products. Winds, air temperature, and specific humidity fields are included in "JRA-55 3-Hourly Model Resolution 2-Dimensional Instantaneous Diagnostic Fields". Precipitation and downward radiation fluxes are found in "JRA-55 3-Hourly Model Resolution 2-Dimensional Average Diagnostic Fields". (Note the difference between instantaneous and averaged data products. There are several JRA55 datasets available, you will likely have to scroll down the page to find these datasets.) Data are also available on a coarser 1.25° grid, but the tools are best used with the native TL319 JRA55 grid.

The fields needed for CICE are

- specific humidity (3-hourly instantaneous), Qa

- temperature (3-hourly instantaneous), Tair
- u-component of wind (3-hourly instantaneous), uatm
- v-component of wind(3-hourly instantaneous), vatm
- downward longwave radiation flux (3 hourly average), flw
- downward solar radiation flux (3 hourly average), fsw
- total precipitation (3 hourly average), fsnow

To customize the dataset for download, choose the "Get a Subset" option. Select the desired times in the "Temporal Selection" section, then click on desired parameters (see list above). After clicking continue, select Output Format "Converted to NetCDF".

Once the data request is made, an email notification will be sent with a dedicated URL that will provide a variety of options for downloading the data remotely. The data will be available to download for 5 days. The raw data consists of multiple files, each containing three months of data for one field.

Data conversion

The script, **configuration/tools/jra55_datasets/interp_jra55_ncdf_bilinear.py**, converts the raw data to CICE input files.

The script uses a bilinear regridding algorithm to regrid from the JRA55 grid to the CICE grid. The scripts use the Python package 'xesmf' to generate bilinear regridding weights, and these regridding weights are written to the file defined by the variable "blin_grid_name" in **interp_jra55_ncdf_bilinear.py**. This filename can be modified by editing **interp_jra55_ncdf_bilinear.py**. The weights file can be re-used if interpolating different data on the same grid. Although not tested in this version of the scripts, additional regridding options are available by xesmf, including 'conservative' and 'nearest neighbor'. These methods have not been tested in the current version of the scripts. The reader is referred to the xESMF web page for further documentation (https://xesmf.readthedocs.io/en/latest/ last accessed 5 NOV 2020).

To use the **interp_jra55_ncdf_bilinear** script, do

```
python3 interp_jra55_ncdf_bilinear.py -h
```

to see the latest interface information

```
usage: interp_jra55_ncdf_bilinear.py [-h] JRADTG gridout ncout

Interpolate JRA55 data to CICE grid

positional arguments:
  JRADTG      JRA55 input file date time group
  gridout     CICE grid file (NetCDF)
  ncout       Output NetCDF filename

optional arguments:
  -h, --help  show this help message and exit
```

Sample usage is

```
./interp_jra55_ncdf_bilinear.py 1996010100_1996033121 grid_gx3.nc JRA55_gx3_03hr_forcing_
↪1996-q1.nc
./interp_jra55_ncdf_bilinear.py 1996040100_1996063021 grid_gx3.nc JRA55_gx3_03hr_forcing_
↪1996-q2.nc
```

(continues on next page)

(continued from previous page)

```
./interp_jra55_ncdf_bilinear.py 1996070100_1996093021 grid_gx3.nc JRA55_gx3_03hr_forcing_
↪1996-q3.nc
./interp_jra55_ncdf_bilinear.py 1996100100_1996123121 grid_gx3.nc JRA55_gx3_03hr_forcing_
↪1996-q4.nc
```

In this case, the 4 quarters of 1996 JRA55 data is going to be interpolated to the gx3 grid. NCO can be used to aggregate these files into a single file

```
ncrcat JRA55_gx3_03hr_forcing_1996-??.nc JRA55_${grid}_03hr_forcing_1996.nc
```

NOTES

- The scripts are designed to read a CICE grid file in netCDF format. This is the "grid_gx3.nc" file above. The NetCDF grid names are hardcoded in **interp_jra55_ncdf_bilinear.py**. If you are using a different grid file with different variable names, this subroutine needs to be updated.
- All files should be placed in a common directory. This includes the raw JRA55 input files, the CICE grid file, and **interp_jra55_ncdf_bilinear.py**. The output files will be written to the same directory.
- The script **configuration/tools/jra55_datasets/make_forcing.csh** was used on the NCAR cheyenne machine in March, 2021 to generate CICE forcing data. It assumes the raw JRA55 is downloaded, but then sets up the python environment, links all the data in a common directory, runs **interp_jra55_ncdf_bilinear.py** and then aggregates the quarterly data using NCO.
- The new forcing files can then be defined in the **ice_in** namelist file using the input variables, `atm_data_type`, `atm_data_format`, `atm_data_dir`, `fyear_init`, and `ycycle`. See *Standalone Forcing* for more information.
- The total precipitation field is mm/day in JRA55. This field is initially read in as snow, but prepare_forcing in **ice_forcing.F90** splits that into rain or snow forcing depending on the air temperature.

4.9 Other things

4.9.1 Running with a Debugger

Availability and usage of interactive debuggers varies across machines. Contact your system administrator for additional information about what's available on your system. To run with an interactive debugger, the following general steps should be taken.

- Setup a case
- Modify the env file and Macros file to add appropriate modules and compiler/ linker flags
- Build the model
- Get interactive hardware resources as needed
- Open a csh shell
- Source the env.${machine} file
- Source cice.settings
- Change directories to the run directory
- Manually launch the executable thru the debugger

4.9.2 Reproducible Sums

Reproducible sums in CICE are set with the namelist *bfbflag*. CICE prognostics results do NOT depend on the global sum implementation when using the default dynamics solver (EVP) or the EAP solver. With these solvers, 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 all dynamics solvers), as well as the model results when using the VP solver. For best performance, the off 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 mostly equivalent to lsum8 (some computations in the VP solver use a different code path when lsum8 is chosen).

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 [41]

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 [14]

4.9.3 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.

4.9.4 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*.

4.9.5 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, water isotopes, and level ice area and volume (from which ridged ice quantities are derived). Salinity, enthalpies, age, aerosols, isotopes, 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

FIVE

INDEX OF PRIMARY VARIABLES AND PARAMETERS

This index defines many (but not all) of the symbols used frequently in the CICE model code. All quantities in the code are expressed in MKS units (temperatures may take either Celsius or Kelvin units). Deprecated parameters are listed at the end.

Namelist variables are partly included here, but they are fully documented in section *Tables of Namelist Options*.

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		
a3Df	history field accumulations, 3D fsd		
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		
a4Df	history field accumulations, 4D categories, fsd		
a_min	minimum area concentration for computing velocity	0.001	
a_rapid_mode	brine channel diameter		
add_mpi_barriers	turns on MPI barriers for communication throttling		
advection	type of advection algorithm used (‘remap’ or ‘upwind’)	remap	
afsd(n)	floe size distribution (in category n)		
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		
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		

continues on next page

Table 1 – continued from previous page

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	
atmiter_conv	convergence criteria for ustar	0.00	
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 (‘similarity’, ‘constant’, or ‘mixed’)		
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			
bfbflag	for bit-for-bit reproducible diagnostics, and reproducible outputs when using the VP solver		
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		
bphi	porosity of ice layers on bio grid		
brlx	relaxation constant for dynamics (momentum)		
bTiz	temperature of ice layers on bio grid		
C			

continues on next page

Table 1 – continued from previous page

c<n>	real(n)		
rotate_wind	if true, rotate wind/stress components to computational grid	T	
calc_dragio	if true, calculate `dragio` from `iceruf_ocn` and `thickness_ocn_layer1`	F	
calc_strair	if true, calculate wind stress	T	
calc_Tsfc	if true, calculate surface temperature	T	
capping	parameter associated with capping method of viscosities	1.0	
capping_method	namelist to specify capping method	hibler	
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	
conserv_check	if true, check conservation		
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	
cpl_frazil	• type of frazil ice coupling		
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		

continues on next page

Table 1 – continued from previous page

cyp	combination of HTE values		
D			
d_afsd_[proc]	change in FSD due to processes		
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	
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	
day_init	the initial day of the month		
dbl_kind	definition of double precision	selected_real_kind(13)	
debug_blocks	write extra diagnostics for blocks and decomposition	.false.	
debug_forcing	write extra diagnostics for forcing inputs	.false.	
debug_model	Logical that controls extended model point debugging.		
debug_model_i	Local i gridpoint that defines debug_model point output.		
debug_model_iblk	Local iblk value that defines debug_model point output.		
debug_model_j	Local j gridpoint that defines debug_model point output.		
debug_model_task	Local mpi task value that defines debug_model point output.		
debug_model_step	Initial timestep for output from the debug_model flag.		
Delta	function of strain rates (see Section *Dynamics*)	1/s	
deltaminEVP	minimum value of Delta for EVP (see Section *Dynamics*)	1/s	
deltaminVP	minimum value of Delta for VP (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	
DminTarea	deltamin * tarea	m^2/s	

continues on next page

Table 1 – continued from previous page

dms	dimethyl sulfide concentration	mmol/m^3	
dmsp	dimethyl sulfoniopropionate concentration	mmol/m^3	
dpscale	time scale for flushing in permeable ice	1×10^{-3}	
drhosdwind	wind compaction factor for snow	27.3 kg s/m^4	
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	Δ temperature per Δ snow grain radius	1. deg	
dte	subcycling time step for EVP dynamics (Δt_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_base	reference date for restart output, zero or init		
dumpfreq_n	restart output frequency		
dump_last	if true, write restart on last time step of simulation		
dwavefreq	widths of wave frequency bins	1/s	
dxE	width of E cell (Δx) through the middle	m	
dxN	width of N cell (Δx) through the middle	m	
dxT	width of T cell (Δx) through the middle	m	
dxU	width of U cell (Δx) through the middle	m	
dxhy	combination of HTE values		
dyE	height of E cell (Δy) through the middle	m	
dyN	height of N cell (Δy) through the middle	m	
dyT	height of T cell (Δy) through the middle	m	
dyU	height of U cell (Δy) through the middle	m	
dyhx	combination of HTN values		
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		
earea	area of E-cell	m^2	
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	
emask	land/boundary mask, T east edge (E-cell)		
emissivity	emissivity of snow and ice	0.985	
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	
etax2	2 x eta (shear viscosity)	kg/s	
evap	evaporative water flux	kg/m^2/s	
ew_boundary_type	type of east-west boundary condition		

continues on next page

Table 1 – continued from previous page

elasticDamp	coefficient for calculating the parameter E, 0< elasticDamp <1	0.36	
e_yieldcurve	yield curve minor/major axis ratio	2	
e_plasticpot	plastic potential minor/major axis ratio	2	
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	
floe_rad_l	lower bounds for FSD size bins (radius)	m	
floe_rad_c	centers of FSD size bins (radius)	m	
floe_binwidth	width of FSD size bins (radius)	m	
flux_bio	all biogeochemistry fluxes passed to ocean		
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	
fmU	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		

continues on next page

Table 1 – continued from previous page

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	
fsloss	rate of snow loss to leads	kg/m^2 s	
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_vdr	visible direct shortwave penetrating to ocean	W/m^2	
fswthru_vdf	visible diffuse shortwave penetrating to ocean	W/m^2	
fswthru_idr	near IR direct shortwave penetrating to ocean	W/m^2	
fswthru_idf	near IR diffuse shortwave penetrating to ocean	W/m^2	
fswthru_ai	grid-box-mean shortwave penetrating to ocean (fswthru)	W/m^2	
fyear	current forcing data year		
fyear_final	last forcing data year		
fyear_init	initial forcing data year		
G			
gravit	gravitational acceleration	9.80616 m/s^2	
grid_atm	grid structure for atm forcing/coupling fields, ‘A’, ‘B’, ‘C’, etc		
grid_atm_dynu	grid for atm dynamic-u forcing/coupling fields, ‘T’, ‘U’, ‘N’, ‘E’		
grid_atm_dynv	grid for atm dynamic-v forcing/coupling fields, ‘T’, ‘U’, ‘N’, ‘E’		
grid_atm_thrm	grid for atm thermodynamic forcing/coupling fields, ‘T’, ‘U’, ‘N’, ‘E’		
grid_file	input file for grid info		
grid_format	format of grid files		
grid_ice	structure of the model ice grid, ‘B’, ‘C’, etc		
grid_ice_dynu	grid for ice dynamic-u model fields, ‘T’, ‘U’, ‘N’, ‘E’		
grid_ice_dynv	grid for ice dynamic-v model fields, ‘T’, ‘U’, ‘N’, ‘E’		
grid_ice_thrm	grid for ice thermodynamic model fields, ‘T’, ‘U’, ‘N’, ‘E’		
grid_ocn	grid structure for ocn forcing/coupling fields, ‘A’, ‘B’, ‘C’, etc		
grid_ocn_dynu	grid for ocn dynamic-u forcing/coupling fields, ‘T’, ‘U’, ‘N’, ‘E’		

continues on next page

Table 1 – continued from previous page

grid_ocn_dynv	grid for ocn dynamic-v forcing/coupling fields, ‘T’, ‘U’, ‘N’, ‘E’		
grid_ocn_thrm	grid for ocn thermodynamic forcing/coupling fields, ‘T’, ‘U’, ‘N’, ‘E’		
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		
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,T,T,T,T	
histfreq	units of history output frequency: y, m, w, d or 1	m,x,x,x,x	
histfreq_base	reference date for history output, zero or init		
histfreq_n	integer output frequency in histfreq units	1,1,1,1,1	
history_chunksize	history chunksizes in x,y directions (_format=’hdf5’ only)	0,0	
history_deflate	compression level for history (_format=’hdf5’ only)	0	
history_dir	path to history output files		
history_file	history output file prefix		
history_format	history file format		
history_iotasks	history output total number of tasks used		
history_precision	history output precision: 4 or 8 byte	4	
history_rearranger	history output io rearranger method		
history_root	history output io root task id		
history_stride	history output io task stride		
hist_time_axis	history file time axis interval location: begin, middle, end	end	
hist_suffix	suffix to *history_file* in filename. x means no suffix	x,x,x,x,x	
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)	m	
hs1	snow depth of transition to pond ice	0.03 m	
hs_ssl	snow surface scattering layer thickness	0.04 m	

continues on next page

Table 1 – continued from previous page

Hstar	determines mean thickness of ridged ice	25. m	
HTE	length of eastern edge (Δy) of T-cell	m	
HTN	length of northern edge (Δx) of T-cell	m	
HTS	length of southern edge (Δx) of T-cell	m	
HTW	length of western edge of (Δy) 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_data_conc	ice initialization concentration, used mainly for box tests		
ice_data_dist	ice initialization distribution, used mainly for box tests		
ice_data_type	ice initialization mask, used mainly for box tests		
ice_hist_field	type for history variables		
ice_ic	choice of initial conditions (see *Ice Initialization*)		
ice_stdout	unit number for standard output		
ice_stderr	unit number for standard error output		
ice_ref_salinity	reference salinity for ice–ocean exchanges		
icells	number of grid cells with specified property (for vectorization)		
iceruf	ice surface roughness at atmosphere interface	$5.\times10^{-4}$ m	
iceruf_ocn	under-ice roughness (at ocean interface)	0.03 m	
iceEmask	dynamics ice extent mask (E-cell)		
iceNmask	dynamics ice extent mask (N-cell)		
iceTmask	dynamics ice extent mask (T-cell)		
iceUmask	dynamics ice extent mask (U-cell)		
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		

continues on next page

Table 1 – continued from previous page

kappav	visible extinction coefficient in ice, wavelength<700nm	1.4 m^{-1}	
kcatbound	category boundary formula		
kdyn	type of dynamics (1 = EVP, 2 = EAP, 3 = VP, 0,-1 = off)	1	
kg_to_g	kg to g conversion factor	1000.	
kice	thermal conductivity of fresh ice ([4])	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		
kmt_type	file, default, channel, wall, or boxislands	file	
krdg_partic	ridging participation function	1	
krdg_redist	ridging redistribution function	1	
krgdn	mean ridge thickness per thickness of ridging ice		
ksno	thermal conductivity of snow	0.30 W/m/deg	
kstrength	ice stength formulation (1= [52], 0 = [15])	1	
ktherm	thermodynamic formulation (-1 = off, 1 = [4], 2 = mushy)		
L			
l_brine	flag for brine pocket effects		
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		
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$\times$10^{-6} m/s deg^{-m2}	
m2	constant for lateral melt rate	1.36	
m2_to_km2	m^2 to km^2 conversion	1$\times$10^{-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		

continues on next page

Table 1 – continued from previous page

max_ntrcr	maximum number of tracers available	5	
maxraft	maximum thickness of ice that rafts	1. m	
mday	model day of the month		
meltb	basal ice melt	m	
meltl	lateral ice melt	m	
melts	snow melt	m	
meltsliq	snow melt mass	kg/m^2	
meltsliqn	snow melt mass in category n	kg/m^2	
meltt	top ice melt	m	
min_salin	threshold for brine pockets	0.1 ppt	
mlt_onset	day of year that surface melt begins		
mmonth	model month number		
monthp	previous month number		
month_init	the initial month		
mps_to_cmpdy	m per s to cm per day conversion	8.64×10^6	
msec	model seconds elasped into day		
mtask	local processor number that writes debugging data		
mu_rdg	e-folding scale of ridged ice		
myear	model year		
myear_max	maximum allowed model year		
my_task	task ID for the current processor		
N			
n_aero	number of aerosol species		
narea	area of N-cell	m^2	
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	
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		
nfreq	number of wave frequency bins	25	
nfsd	number of floe size categories	12	
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	7	
nit	nitrate concentration	mmol/m^3	
nlt_bgc_[chem]	ocean sources and sinks for biogeochemistry		
nmask	land/boundary mask, T north edge (N-cell)		

continues on next page

Table 1 – continued from previous page

nml_filename	namelist file name		
nprocs	total number of processors		
npt	total run length values associate with npt_unit		
npt_unit	units of the run length, number set by npt		
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		
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		
ocn_data_type	source of surface temperature, salinity data		
omega	angular velocity of Earth	7.292×10^{-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		

continues on next page

Table 1 – continued from previous page

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	π		
pi2	2π		
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		
Pstar	ice strength parameter	2.75×10^4N/m^2	
puny	a small positive number	1×10^{-11}	
Q			
Qa	specific humidity at 10 m	kg/kg	
qdp	deep ocean heat flux	W/m^2	
qqqice	for saturated specific humidity over ice	1.16378×10^7kg/m^3	
qqqocn	for saturated specific humidity over ocean	6.275724×10^6kg/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	

continues on next page

Table 1 – continued from previous page

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	6.37×10^6 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	
rep_prs	replacement pressure	N/m	
revp	real(revised_evp)		
restart	if true, initialize ice state from file	T	
restart_age	if true, read age restart file		
restart_bgc	if true, read bgc restart file		
restart_chunksize	restart chunksizes in x,y directions (_format='hdf5' only)	0,0	
restart_deflate	compression level for restart (_format='hdf5' only)	0	
restart_dir	path to restart/dump files		
restart_file	restart file prefix		
restart_format	restart file format		
restart_iotasks	restart output total number of tasks used		
restart_rearranger	restart output io rearranger method		
restart_root	restart output io root task id		
restart_stride	restart output io task stride		
restart_[tracer]	if true, read tracer restart file		
restart_ext	if true, read/write halo cells in restart file		
restart_coszen	if true, read/write coszen 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	
rhos_cmp	density of snow due to wind compaction	kg/m^3	
rhos_cnt	density of ice and liquid content of snow	kg/m^3	
rhosi	average sea ice density (for hbrine tracer)	940. kg/m^3	

continues on next page

Table 1 – continued from previous page

rhosmax	maximum snow density	450 kg/m^3	
rhosmin	minimum snow density	100 kg/m^3	
rhosnew	new snow density	100 kg/m^3	
rhow	density of seawater	1026. kg/m^3	
rnilyr	real(nlyr)		
rside	fraction of ice that melts laterally		
rsnw	snow grain radius	10^{-6} m	
rsnw_fall	freshly fallen snow grain radius	100. $\times$ 10^{-6} m	
rsnw_mlt	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	
rsnw_tmax	maximum snow radius	1500. $\times$ 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 ([4])	3.2 ppt	
scale_factor	scaling factor for shortwave radiation components		
seabed_stress	if true, calculate seabed stress	F	
seabed_stress_method	method for calculating seabed stress (‘LKD’ or ‘probabilistic’)	LKD	
secday	number of seconds in a day	86400.	
sec_init	the initial second		
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’ or ‘dEdd_snicar_ad’)		
sig1(2)	principal stress components $\sigma_{n,1}$, $\sigma_{n,2}$ (diagnostic)		
sigP	internal ice pressure	N/m	
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	

continues on next page

Table 1 – continued from previous page

snoice	snow–ice formation	m	
snowpatch	length scale for parameterizing nonuniform snow coverage	0.02 m	
skl_bgc	biogeochemistry on/off		
smassice	mass of ice in snow from smice tracer	kg/m^2	
smassliq	mass of liquid in snow from smliq tracer	kg/m^2	
snowage_drdt0	initial rate of change of effective snow radius		
snowage_rhos	snow aging parameter (density)		
snowage_kappa	snow aging best-fit parameter		
snowage_tau	snow aging best-fit parameter		
snowage_T	snow aging parameter (temperature)		
snowage_Tgrd	snow aging parameter (temperature gradient)		
snw_aging_table	snow aging lookup table		
snw_filename	snowtable filename		
snw_tau_fname	snowtable file tau fieldname		
snw_kappa_fname	snowtable file kappa fieldname		
snw_drdt0_fname	snowtable file drdt0 fieldname		
snw_rhos_fname	snowtable file rhos fieldname		
snw_Tgrd_fname	snowtable file Tgrd fieldname		
snw_T_fname	snowtable file T fieldname		
snwgrain	activate snow metamorphosis		
snwlvlfac	fractional increase in snow depth for redistribution on ridges	0.3	
snwredist	type of snow redistribution		
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	5.67×10^{-8} W/m^{2}K^4	
stop_now	if 1, end program execution		
strairx(y)U	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	N/m	
stress12	internal ice stress, σ_{12}	N/m	
stressm	internal ice stress, $\sigma_{11} - \sigma_{22}$ (σ_2 in the doc)	N/m	
stressp	internal ice stress, $\sigma_{11} + \sigma_{22}$ (σ_1 in the doc)	N/m	
strintx(y)U	divergence of internal ice stress, x(y)	N/m^2	
strocnx(y)U	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)U	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	

continues on next page

Table 1 – continued from previous page

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		
Tf	freezing temperature	C	
Tffresh	freezing temp of fresh ice	273.15 K	
tfrz_option	form of ocean freezing temperature		
saltflux_option	form of coupled salt flux		
thinS	minimum ice thickness for brine tracer		
timer_stats	logical to turn on extra timer statistics	.false.	
timesecs	total elapsed time in seconds	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	
Tinz	Internal ice temperature	C	
TLAT	latitude of cell center	radians	
Tliquidus_max	maximum liquidus temperature of mush	0. C	
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		
tr_iage	if true, use ice age tracer		
tr_lvl	if true, use level ice area and volume tracers		
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.$\times$10^{-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	

continues on next page

Table 1 – continued from previous page

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 corner (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		
use_smliq_pnd	use liquid in snow for ponds		
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	1.79×10^{-3} kg/m/s	
visc_method	method for calculating viscosities ('avg_strength' or 'avg_zeta')	avg_zeta	
vocn	ocean current in the y-direction	m/s	
vonkar	von Karman constant	0.4	
vort	vorticity	1/s	
vraftn	volume of rafted ice	m	
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	
wave_sig_ht	significant height of waves	m	
wave_spectrum	wave spectrum	m^2/s	
wavefreq	wave frequencies	1/s	
wind	wind speed	m/s	
windmin	minimum wind speed to compact snow	10 m/s	

continues on next page

Table 1 – continued from previous page

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, computed in the model calendar		
yield_curve	type of yield curve	ellipse	
yieldstress11(12, 22)	yield stress tensor components		
year_init	the initial year		
Z			
zetax2	2 x zeta (bulk viscosity)	kg/s	
zlvl	atmospheric level height (momentum)	m	
zlvs	atmospheric level height (scalars)	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	
Deprecated options and parameters			
heat_capacity	if true, use salinity-dependent thermodynamics	T	
kseaice	thermal conductivity of ice for zero-layer thermodynamics	2.0 W/m/deg	
ktherm	thermodynamic formulation (0 = zero-layer, 1 = [4], 2 = mushy)		
tr_pond_cesm	if true, use CESM melt pond scheme		

CHAPTER

SIX

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] A. Arakawa and V.R. Lamb. Computational design of the basic dynamical processes of the ucla general circulation model. In Julius Chang, editor, *General Circulation Models of the Atmosphere*, volume 17 of Methods in Computational Physics: Advances in Research and Applications, pages 173–265. Elsevier, 1977. URL: https://www.sciencedirect.com/science/article/pii/B9780124608177500094, doi:https://doi.org/10.1016/B978-0-12-460817-7.50009-4.

[3] 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.

[4] 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.

[5] Amelie Bouchat, Nils Hutter, Jerome Chanut, Frederic Dupont, Dmitry Dukhovskoy, Gilles Garric, Younjoo J. Lee, Jean-Francois Lemieux, Camille Lique, Martin Losch, Wieslaw Maslowski, Paul G. Myers, Einar Olason, Pierre Rampal, Till Rasmussen, Claude Talandier, Bruno Tremblay, and Qiang Wang. Sea ice rheology experiment (sirex): 1. scaling and statistical properties of sea-ice deformation fields. *Journal of Geophysical Research: Oceans*, 127(4):e2021JC017667, 2022. URL: https://agupubs.onlinelibrary.wiley.com/doi/abs/10.1029/2021JC017667, doi:https://doi.org/10.1029/2021JC017667.

[6] 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.

[7] S. Bouillon, M.A Morales Maqueda, V. Legat, and T. Fichefet. An elastic-viscous-plastic sea ice model formulated on Arakawa B and C grids. *Ocean Modelling*, 27:174–184, 2009. URL: doi:10.1016/j.ocemod.2009.01.004.

[8] 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.

[9] 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.

[10] 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.

[11] F. Dupont, D. Dumont, J.F. Lemieux, E. Dumas-Lefebvre, and A. Caya. A probabilistic seabed-ice keel interaction model. *The Cryosphere*, 16:1963–1977, 2022. URL: https://doi.org/10.5194/tc-16-1963-2022.

[12] 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.

[13] 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.

[14] 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.

[15] 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.

[16] 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.

[17] 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.

[18] 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.

[19] C. Horvat and E. Tziperman. A prognostic model of the sea-ice floe size and thickness distribution. *The Cryosphere*, 9(6):2119–2134, 2015. URL: http://dx.doi.org/10.5194/tc-9-2119-2015.

[20] 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.

[21] 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.

[22] 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.

[23] 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/main/doc/PDF/LAUR-03-2219.pdf.

[24] 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.

[25] 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.

[26] 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/main/doc/PDF/JDWSTWLG06.pdf.

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

[28] 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.

[29] M. Kimmritz, S. Danilov, and M. Losch. The adaptive EVP method for solving the sea ice momentum equation. *Ocean Modelling*, 101:59–67, 2016. URL: http://dx.doi.org/10.1016/j.ocemod.2016.03.004.

[30] N.V. Koldunov, S. Danilov, D. Sidorenko, N. Hutter, M. Losch, H. Goessling, N. Rakowsky, P. Scholz, D. Sein, Q. Wang, and T. Jung. Fast EVP solutions in a high-resolution sea ice model. *Journal of Advances in Modeling Earth Systems*, 11(5):1269–1284, 2019. URL: http://doi.wiley.com/10.1029/2018MS001485.

[31] M. Kreyscher, M. Harder, P. Lemke, and G.M. Flato. Results of the Sea Ice Model Intercomparison Project: evaluation of sea ice rheology schemes for use in climate simulations. *J. Geophys. Res.*, 105(C5):11299–11320, 2000.

[32] D. Lavoie, K. Denman, and C. Michel. Modeling ice algal growth and decline in a seasonally ice- covered region of the Arctic (Resolute Passage, Canadian Archipelago). *J. Geophys. Res. Oceans*, 2005. URL: http://dx.doi.org/10.1029/2005JC002922.

[33] J.-F. Lemieux, B. Tremblay, S. Thomas, J. Sedláček, and L. A. Mysak. Using the preconditioned Generalized Minimum RESidual (GMRES) method to solve the sea-ice momentum equation. *J. Geophys. Res. Oceans*, 113(C10):, 2008. URL: https://agupubs.onlinelibrary.wiley.com/doi/abs/10.1029/2007JC004680, arXiv:https://agupubs.onlinelibrary.wiley.com/doi/pdf/10.1029/2007JC004680, doi:10.1029/2007JC004680.

[34] 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.

[35] 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.

[36] 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.

[37] 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.

[38] 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.

[39] 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.

[40] 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.

[41] 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.

[42] 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.

[43] 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.

[44] 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.

[45] 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.

[46] D. Ringeisen, L.B. Tremblay, and M. Losch. Non-normal flow rules affect fracture angles in sea ice viscous-plastic rheologies. *The Cryosphere*, 15:2873–2888, 2021. URL: https://doi.org/10.5194/tc-15-2873-2021.

[47] L. A. Roach, C. Horvat, S. M. Dean, and C. M. Bitz. An emergent sea ice floe size distribution in a global coupled ocean-sea ice model. *J. Geophys. Res. Oceans*, 123(6):4322–4337, 2018. URL: http://dx.doi.org/10.1029/2017JC013692.

[48] L.A. Roach, C. M. Bitz, C. Horvat, and S. M. Dean. Advances in modelling interactions between sea ice and ocean surface waves. *Journal of Advances in Modeling Earth Systems*, 2019. URL: http://doi.wiley.com/10.1029/2019MS001836.

[49] 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.

[50] 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.

[51] 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.

[52] 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.

[53] Y. Saad. A Flexible Inner-Outer Preconditioned GMRES Algorithm. *SIAM J. Sci. Comput.*, 14(2):461–469, 1993. URL: https://doi.org/10.1137/0914028, doi:10.1137/0914028.

[54] 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.

[55] 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/main/doc/PDF/LAUR-95-1146.pdf.

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

[57] A. Tagliabue, L. Bopp, and O. Aumont. Evaluating the importance of atmospheric and sedimentary iron sources to Southern Ocean biogeochemistry. *Geophys. Res. Lett.*, 2009. URL: http://dx.doi.org/10.1029/2009GL038914.

[58] 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.

[59] 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.

[60] 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.

[61] H. Tsujino, S. Urakawa, R.J. Small, W.M. Kim, S.G. Yeager, and et al. JRA-55 based surface dataset for driving ocean–sea-ice models (JRA55-do). *Ocean Modelling*, 130:79–139, 2018. URL: http://dx.doi.org/10.1016/j.ocemod.2018.07.002.

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

[63] 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.

[64] 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.

[65] 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.

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

[67] 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.

[68] 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.