MITgcm SEAICE manual

Tim Hill

August 17, 2018

Contents

1 Introduction

This document aims to serve as a starting point for running the MITgcm in configurations with ice. I have the model at a functional point and hope to leave some resources so it is easy to pick up where I left off.

The first section serves as an introduction to MITgcm and should be a good resource to carry out the first few functional model runs, including running with the ice package.

The next few sections outline some specific things I learned about the model, and may be useful to someone trying to extend the model past what is outlined here. The tools I created for visualizing results are described. Some specific model configurations that did not work are explained, and problems I encountered with the model or with the Graham environment are discussed.

2 Setting up the MITgcm

The most useful references for the MITgcm are:

- The fluids wiki has a wealth iof information about computing resources, running on the SHARCNET cluster, and has an MITgcm tutorial (discussed later).
- The MITgcm homepage
- The new MITgcm documentation is on read the docs, but is incomplete at the time of writing. The old documentation is still accessible but has some unfortunate qualities, including embedding tabels as images so they are not searchable.
- The MITgcm github repo is where I usually look at and search through the model code if I need to.

2.1 Basic tutorial

The Fluids wiki tutorial is a good tutorial to start with the model. The tutorial was originally written by Emily Tyhurst, and I have corrected a few mistakes in the tutorial.

The tutorial is largely self-contained, but I want to say a word about directory structure. MITgcm expects your case files to be contained in the same directory as the model. For example, if you make a folder called

`simulation1` for your simulation, this folder might be in the same directory as the MITgcm directories `doc/`, `eesupp/`, `pkg/`, `model/`, etc., in which case the directory structure would look like

```
/home/user/MITgcm

      simulation1/

      doc/
      eesupp
      jobs/
      lsopt/
      model/
      optim/
      pkg/
      tools/
      utils/
      verification/
      ...
```

I had a folder I called `MITgcmdata/` which was a git repo that contained all my model runs. Then my directory tree was

```
/home/user/MITgcm

      MITgcmdata/
          simulation1
          simulation2
          ...

      doc/
      eesupp
      jobs/
      lsopt/
      model/
      optim/
      pkg/
      tools/
      utils/
      verification/
      ...
```

Having a structure like this makes compiling the model much easier. You can in theory compile the model from anywhere as long as you point the `genmake2` script to the model directories, but in practice this has been harder than expected.

2.2 SEAICE tutorial

The SEAICE package **??** provides a dynamic and thermodynamic sea-ice model for the MITgcm. This package is general purpose enough to model freshwater and ocean ice. The package usually does a good job (at least qualitatively) of recreating dynamics and ice quantities reasonably physically.

Write the rest of this up for small processor numbers (ie HOOD)

3 More details about running MITgcm

This section tries to describe aspects of running the MITgcm in more detail than the previous section. Hopefully this will be a good resource when trying to extend the model past what is included in the tutorials.

MITgcm is structured as a core model + packages. The only packages within my scope are the seaice, external forcing, calendar, and diagnostic packages. In the first subsection I describe each of these packages and try to give some guidance on parameters to choose. The final subsection generally describes more features and parameters which are important to consider: equations of state, checkpoint files, binary input/output, and parameters for the core model.

3.1 Available packages

This section descibres the additional packages available, in particular the SEAICE, EXF/CAL, and DIAGNOSTICS packages.

3.1.1 DIAGNOSTICS

The diagnostics package exists to make it easier to control the output files generated by the model. It is easy to choose which fields are outputted and how frequently files are saved. As usual, the package requires compile-time and run-time options.

Compile time options

At compile time (probably in your `code` directory), you should have the file `DIAGNOSTICS_SIZE.h` (example below). This can often be copied from one of the `verification/` example experiments.

```
C     Diagnostics Array Dimension
C     ---------------------------
C     ndiagMax   :: maximum total number of available diagnostics
C     numlists   :: maximum number of diagnostics list (in data.diagnostics)
C     numperlist :: maximum number of active diagnostics per list (data.
     diagnostics)
C     numLevels  :: maximum number of levels to write    (data.diagnostics)
C     numdiags   :: maximum size of the storage array for active 2D/3D
     diagnostics
C     nRegions   :: maximum number of regions (statistics-diagnostics)
C     sizRegMsk  :: maximum size of the regional-mask (statistics-diagnostics)
C     nStats     :: maximum number of statistics (e.g.: aver,min,max ...)
C     diagSt_size:: maximum size of the storage array for statistics-diagnostics
C Note : may need to increase "numdiags" when using several 2D/3D diagnostics,
C  and "diagSt_size" (statistics-diags) since values here are deliberately small.
      INTEGER    ndiagMax
      INTEGER    numlists, numperlist, numLevels
      INTEGER    numdiags
      INTEGER    nRegions, sizRegMsk, nStats
      INTEGER    diagSt_size
```

```
19       PARAMETER( ndiagMax = 500 )
20       PARAMETER( numlists = 25, numperlist = 50, numLevels=2*Nr )
21       PARAMETER( numdiags = 25*Nr )
22       PARAMETER( nRegions = 0 , sizRegMsk = 1 , nStats = 4 )
23       PARAMETER( diagSt_size = 10*Nr )
24
25
26 CEH3 ;;; Local Variables: ***
27 CEH3 ;;; mode:fortran ***
28 CEH3 ;;; End: ***
```

Listing 1: Example `DIAGNOSTICS_SIZE.h` file

Parameter `numlists` (line 20) and `numdiags` (line 21) can be adjusted if you need more diagnostics.

Run time options

The diagnostics you want to output, and how frequently to write files are specified at runtime in file `data.diagnostics`. Specify the field names in `field` and `filename` lines, and set the frequency. See the following example

```
1  # Diagnostic Package Choices
2  #-----------------
3  # for each output-stream:
4  #  filename(n) : prefix of the output file name (only 8.c long) for outp.stream n
5  #  frequency(n):< 0 : write snap-shot output every |frequency| seconds
6  #               > 0 : write time-average output every frequency seconds
7  #  timePhase(n)     : write at time = timePhase + multiple of |frequency|
8  #  averagingFreq(n) : frequency (in s) for periodic averaging interval
9  #  averagingPhase(n): phase     (in s) for periodic averaging interval
10 #  repeatCycle(n)   : number of averaging intervals in 1 cycle
11 #  levels(:,n) : list of levels to write to file (Notes: declared as REAL)
12 #                 when this entry is missing, select all common levels of this
       list
13 #  fields(:,n) : list of diagnostics fields (8.c) (see "available_diagnostics.log
       "
14 #                 file for the list of all available diag. in this particular
       config)
15 #-----------------
16  &DIAGNOSTICS_LIST
17 # diag_mnc     = .FALSE.,
18 # dumpAtLast   = .TRUE.,
19 #=====================================
20   frequency(1) = -3600.0,
21   timePhase(1) = 0,
22   fields(1, 1) = 'THETA',
23   filename( 1) = 'T',
24 #-----------------
25   frequency(2) = -3600.0,
26   timePhase(2) = 0,
27   fields(1, 2) = 'UVEL',
28   filename( 2) = 'U',
29 #-----------------
30   &
31 ...
```

Listing 2: Example `data.diagnostics` file

For a real model run we would likely want to include more diagnostics by continuing to add to the list in the same way. The full list of diagnostics is listed in the documentation for each package, as well as in the `available_diagnostics.log` file created when you run with the diagnostics package. See the examples in the tutorials for a full working example.

3.2 Core model parameters and recommendations

This section discusses choosing an equation of state, using checkpoint files, time-stepping, binary input/output, and other parameters for the core model.

4 What didn't work

4.1 Advection schemes

4.2 Surface winds

5 Results: cases studied

6 Recurring issues

6.1 NetCDF/HDF5 libraries

6.2 MITgcm python utils

7 Physics of the model

The model has relatively standard physical parameterizations that are usually documented to some extent in the MITgcm documentation. As part of getting the model up and running, I looked into some of the parameterizations and compared to what we expect.

7.1 Radiative cooling

We looked into the radiative cooling of ice when starting to compare MITgcm ice growth rates to the 1-D model CLIMo (Canadian Lake Ice Model).

The radiative cooling should follow the Stefan-Boltzmann law for total radiative power M

$$M = \varepsilon \sigma T^4 \tag{1}$$

Where M is the radiant emittance ($\mathrm{W\,m^{-2}}$), ε is the emissivity, and $\sigma = 5.670\,373 \times 10^{-8}\,\mathrm{W\,m^{-2}\,K^{-4}}$ is the Stefan-Boltzmann constant.

The peak wavelength of the spectral distribution should follow Wien's law

$$\lambda_{max} = \frac{b}{T} \tag{2}$$

Where $b = 2.898 \times 10^{-3}\,\mathrm{m\,K}$ is Wien's displacement constant and T is the temperature. For $0\,^{\circ}\mathrm{C}$, $\lambda_{max} \approx 10^{-5}\,\mathrm{m}$ is deep in the infrared (longwave).

Therefore, all the radiant emittance should be captured as upward longwave radiation.

The emittance will depend on the value of emissivity used. The SEAICE packages takes in ice emissivity with the variable `SEAICE_emissivity` which is actually $\sigma\varepsilon$. The default value is $\sigma\varepsilon = 5.50 \times 10^{-8}\,\mathrm{W\,m^{-2}\,K^{-4}}$. Computing the expected emittance,

$$M = 306.17\,\mathrm{W\,m^{-2}}$$

The model shows very good agreement with this value (fig. **??**)

7.2 Water surface energy budget

As one way to compare the effect of parameters and meteorological forcing values, we compute the total internal energy in heat of the water, as well as the heat fluxes across the water boundary. We expect that

$$\frac{dQ}{dt} = \Phi_{net} = \Phi_{ice} + \Phi_{atm} \tag{3}$$

The model outputs $-\Phi_{net}$ as the diagnostic `SIqnet` (heat flux from the water into ice if ice covered, atmosphere if open water). To check this, we compute the internal energy of the water as

$$Q = c_w \sum_{x,y,z} \rho \Delta V \Delta T \tag{4}$$

Where the sum is over all grid cells, ΔV is the volume of each cell, and ΔT is the temperature at time t minus the initial temperature.

The derivative is taken using the most basic scheme,

$$\frac{dQ}{dt} \approx \frac{Q(t_{n+1}) - Q(t_n)}{t_{n+1} - t_n}$$

Finally, a plot of $\frac{dQ}{dt} + $ `SIqnet` should be near zero (not identically zero because of the derivative scheme) for all times. We have found this to be the case.

7.3 Surface albedo

In comparing the MITgcm ice growth rate to CLIMo, I investigated the albedo parameterization of MITgcm. The model classifies ice according to the temperature of the surface water T_{surf} comapred to the freezing/melting temperature T_{melt}

$$\text{ice} = \begin{cases} \text{dry}, & T_{surf} < T_{melt} \\ \text{wet}, & T_{surf} \geq T_{melt} \end{cases}$$

The default ice albedo is

$$\alpha_i = \begin{cases} 0.75, T_{surf} < T_{melt} \\ 0.66, T_{surf} \geq T_{melt} \end{cases}$$

The default snow albedo is

$$\alpha_s = \begin{cases} 0.84, T_{surf} < T_{melt} \\ 0.70, T_{surf} \geq T_{melt} \end{cases}$$

And the values are controlled by the input variables `SEAICE_dryIceAlb`, `SEAICE_wetIceAlb`, `SEAICE_drySnowAlb`, and `SEAICE_wetSnowAlb`.

7.4 Thermodynamics

The SEAICE package is mainly intended for ice dynamics, but includes basic thermodynamics. The THSICE package is be fully compatible with SEAICE and include improved thermodynamics. However, the THSICE package is not well suited for freshwater simulations, so for the purpose of this document it is recommended to continue to use only the SEAICE package for dynamics and thermodynamics.

The THSICE package is intended to improve the thermodynamics of saltwater ice. When ice forms over salt water, some pockets of salt water called brine pockets get trapped in the ice. These brine pockets provide interesting thermodynamics. When the ice is warmed, some of the ice around the brine pockets will be melted. This meltwater must be freshwater, so it dilutes the salt in the brine pocket. This can sufficiently dilute the brine pocket so that the pocket freezes completely, releasing latent heat to the surrounding ice [**?**].

Clearly this process is irrelevant for freshwater lake ice. While the three-layer formulation of [**?**] is an improvement over what is described in the SEAICE documentation, the implementation does not work for freshwater. The model tends to NaN once ice starts to form. Moreover, the SEAICE documentation says "NOTE: THIS SECTION IS TERRIBLY OUT OF DATE" about the thermodynamics description in the documentation. Therefore, we can't really compare the packages anyways.

7.5 Total kinetic energy

I compute the total kinetic energy of the water as an additional way to compare the differences between different parameters and configurations. The total kinetic energy is computed as

$$\text{TKE} = \frac{1}{2} \sum_{x,y,z} \rho \left(u^2 + v^2 + w^2 \right) \Delta V \tag{5}$$

Where $\rho = \rho_0 + \rho'$ is the total (reference + anomaly) density of the gri cell, u, v, w are the x, y, z velocities, and ΔV is the volume of the grid cell.

7.6 Dynamical ice model

The equation governing the dynamical evolution of the ice is the conservation of momentum equation

$$m \frac{D\mathbf{u}}{Dt} = -mf\mathbf{k} \times \mathbf{u} - m\nabla\phi(0) + \mathbf{F} + \tau_{\mathbf{air}} + \tau_{\mathbf{ocean}} \tag{6}$$

Where $m = m_i + m_s$ is the total mass of ice and snow per unit area, $\phi(0)$ is the sea surface height potential, $\mathbf{F} = \nabla \cdot \sigma$ is the divergence of the ice stress tensor, and τ_{air}, τ_{ocean} are the wind-ice and ocean-ice stresses. The wind and ocean stresses are parameterized as

$$\begin{aligned} \tau_{\mathbf{air}} &= \rho_{air} C_{air} \left\| \mathbf{U}_{air} - \mathbf{u} \right\| R_{air} \left(\mathbf{U}_{air} - \mathbf{u} \right) \\ \tau_{\mathbf{water}} &= \rho_{water} C_{water} \left\| \mathbf{U}_{water} - \mathbf{u} \right\| R_{water} \left(\mathbf{U}_{water} - \mathbf{u} \right) \end{aligned} \tag{7}$$

Where $\mathbf{U_i}$ are the surface winds of the atmosphere and water, C_i are air and water drag coefficients, ρ_i are reference densities, and R_i are rotation matrices.