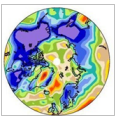


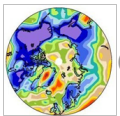
Modifying Model Code

Andrew Gettelman





What we have done:

- Log-in to a super-computer
- Run the model
- Change run options and output
- Run the diagnostic scripts
- Change configuration options
- Run the Single Column Model
- Visualize output with NCVIEW & NCL

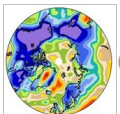


The Choice

Blue Pill: 
Leave now and use CAM as
before without modifying code

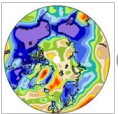
Red Pill: 
“You stay in Wonderland and I
show you how deep the rabbit
hole goes”

Morpheus, 1999



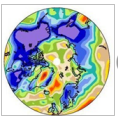
Modifying Model Code

- How to find things in the code: `grepccm`
- Tracking Changes in Source code
 - Good practices
- Simple modifications to source code
- Software interface standards
- Exercises: Simple code modifications
 - `grepccm`
 - Adding an output field from a variable
 - Modifying a parameter in the code
 - Advanced: Make a new variable



Structure of CAM Code

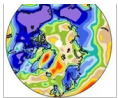
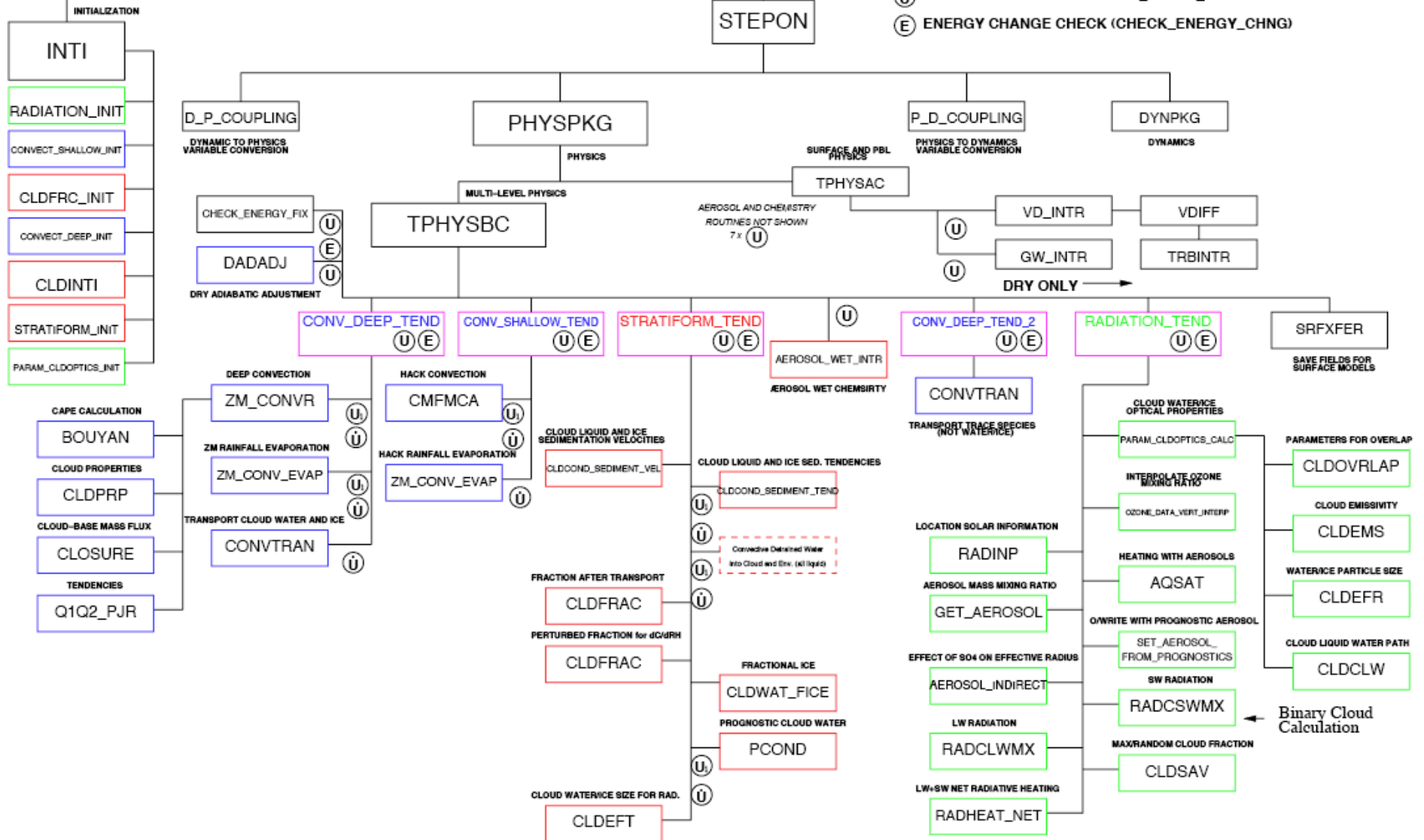
- CAM code (particularly physics code) is typically divided into 'modules' that perform a set of tasks and match a standard interface
- Modules typically have the following components:
 - Register :tell the model how to set up
 - Initialize :set up constants
 - Time-step Initialize :specific constants for a timestep
 - Tendency :code run every timestep



MOIST CONVECTION
STRATIFORM
RADIATION
INTERFACE

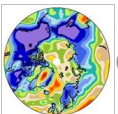
CAM3.0.11

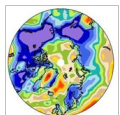
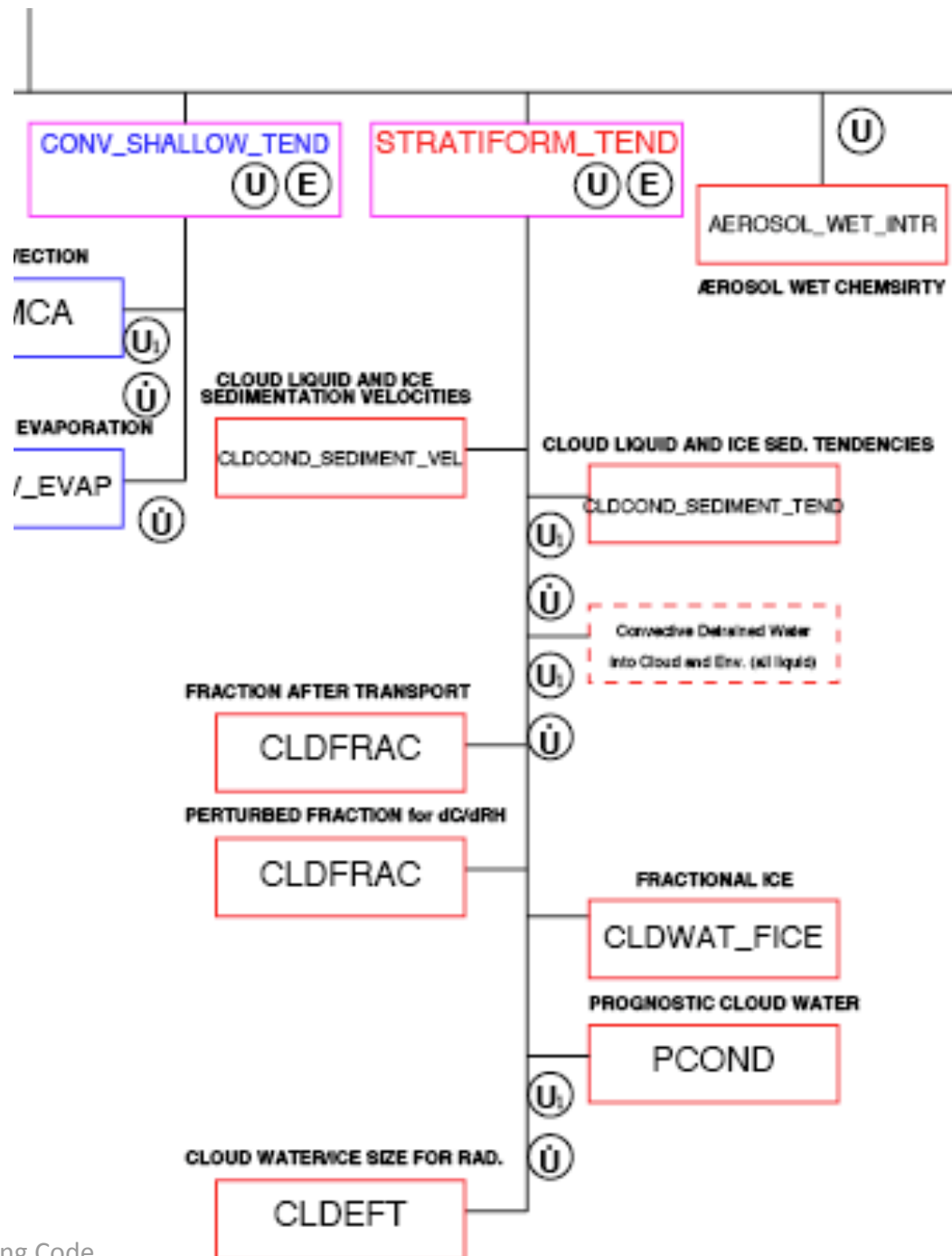
- ⓪ PHYSICS STATE UPDATE (PHYSICS_UPDATE)
- Ⓛ PHYSICS COPY STATE UPDATE (PHYSICS_UPDATE)
- Ⓜ TENDENCY UPDATE (PHYSICS_PTEND_SUM)
- Ⓨ ENERGY CHANGE CHECK (CHECK_ENERGY_CHNG)



CAM Physics Structure

- Note on the following the different components of 'stratiform' (stratiform.F90)
 - stratiform_init
 - stratiform_tend
- It then calls a bunch of other routines too:
 - cldcond_sediment_vel
 - cldcon_Sediment_tend
 - cldfrc
 - cldwat_fice
 - ccond
- Some routines are in stratiform.F90
- Some are in other modules (e.g.: cloud_fraction.F90)
- Similar for others (e.g. Radiation)

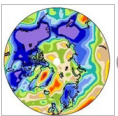




Goal: Standard Interfaces

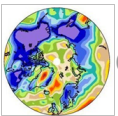
With the right coding, other process codes can usually be integrated into CAM using an ‘interface’:

- ‘Interface’ translates the CAM state into definitions needed by a parameterization
- Takes the result, converts it to a tendency, and makes sure it gets applied to CAM state
- This is how most physics parameterizations work in CAM



Tendencies: $x_1 = x_0 + dX/dt * \Delta t$

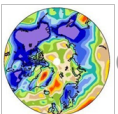
- Logical for an 'Interface' Model
- Easy to process split or time split
- Easy to check energy and mass conservation
 - Minimizes and centralizes impact on model
- Processes flexible with time-step
- Easier to diagnose processes and close budgets
 - Tendencies can often be written out as variables



grepccm: how to find things

grepccm is a modified version of grep that from a build directory (“bld”) will search for a string in all CAM source code directories

grepccm keys off of ‘Filepath’: a file created when the model is built (compiled)

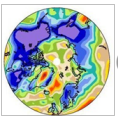


Example: grepccm

- grepccm sst_option (used for aquaplanet)

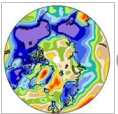
- Result:

```
---- searching /fis01/cgd/cms/andrew/camtutorial2009/  
   cam_tutorial_cam3_6_48/models/ocn/dom  
sst_data.F90: integer,parameter :: sst_option = 1  
sst_data.F90: if(sst_option .lt. 1 .or. sst_option .gt. 10) then  
sst_data.F90:   call endrun ('SSTINT: sst_option must be between 1 and  
   10')  
sst_data.F90: if(sst_option == 1 .or. sst_option == 6 .or. &  
sst_data.F90:   sst_option == 7 .or. sst_option == 8   ) then  
sst_data.F90: if(sst_option == 2) then  
sst_data.F90: if(sst_option == 3) then  
sst_data.F90: if(sst_option == 4) then  
sst_data.F90: if(sst_option == 5) then  
sst_data.F90: if(sst_option == 6) then  
sst_data.F90: if(sst_option == 7) then  
sst_data.F90: if(sst_option == 8) then  
sst_data.F90: if(sst_option == 9) then  
sst_data.F90: if(sst_option == 10) then
```



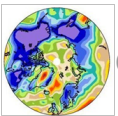
Principles for modifying code

- Track all your changes and all your work
 - Easy reproduceability is key
- Good practice to mark your fortran changes
 - I typically bracket mine with `!++ag`, `!--ag`
- With a script and the modified source code, you can reproduce a run
 - Code takes a 'mod_src' directory and a script
 - Keep these around for EVERY RUN, even minor changes.
 - Good practice: script with same name as case



Principles (2)

- Also DOCUMENT what each run does!
 - case name
 - comments on what you did & why (hypothesis)
 - also basic result or conclusion
- Can be a simple text file
- Can also be a spreadsheet, database, wiki, etc
 - collaborative spaces good for group work if others need access to what you did



Here is a sample: /blhome/andrew/tutorial/tutorial_runs.txt

22 July 2009

test.csh: first test run with regular script (based on run-ibm-tutorial.csh)

23 July 2009

scamtest.csh: test run with scam script (run-scam-tutorial.csh)

24 July 2009

scamschedule.csh

testschdeule.csh

pair of scheduled runs to test bluefire queuing system

test_icritc20ppm.csh

CAM run to test parameter change icritc=20ppm (from 9.5)

test_icritc5ppm.csh

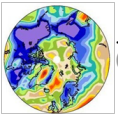
CAM run to test parameter change icritc=5ppm (from 9.5)

25 July 2009

test_ideal.csh

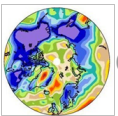
CAM Tutorial: Modifying Code

Ideal physics run (-phys ideal in configure)



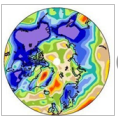
BACKUPS

- Finally: BACK UP YOUR SCRIPTS and CODE!
- `/ptmp` is scrubbed 'occasionally'
 - Do NOT leave code here, only things that can be reproduced (model output)
- `/blhome` is backed up: but don't count on it
- Best to occasionally tar up scripts and back them up.
 - Scripts and source mods are small
- A Backup is part of the exercise. Do it.



Simple Code Modifications

- One common thing is to output a variable that is not already output from the model
- Example: If you look in the documentation, there are fields for in-cloud water path: ICLDIWP (ice) and ICLDTWP (liquid + ice)
- There is no field for ICLDLWP
- Lets make one



What makes an output variable?

- ICLDIWP only appears 3 times:

```
call addfld ('ICLDIWP', 'gram/m2', pver,'A','In-cloud ice water path', &  
  phys_decomp, sampling_seq='rad_lwsw')  
call add_default ('ICLDIWP', 1, ' ' )  
call outfld('ICLDIWP' ,cicewp , pcols,lchnk)
```

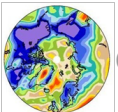
addfld: assigns the variable (init)

add_default: adds it by default to h0 (init)

outfld: tells the code to write a variable (cwp) to it (tend)

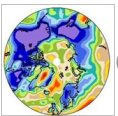
- Not going to make it default. Just need:
 - addfld (init), outfld (tend)
 - variable to put into outfld

- We will do this as an exercise in CAM



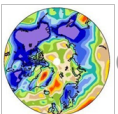
Modifying Parameters in Code

- can change ‘answers’ through the `namelist`
 - CO₂ for example
- or `configure`
 - change the dynamical core or resolution
- Now: changes by modifications to code
 - Fortran code changes
- Start with ‘parameter’ changes



Parameter Adjustment

- What to modify?
- Not a physical constant (gravity, pi, etc)
 - Will discuss this in a minute
- Could change solar constant! (we recently did)
 - That is another story
- Let's pick something unconstrained:
 - Critical mass for auto-conversion of ice to snow:
`icritc`
 - Tomorrow you will do this again in CCSM

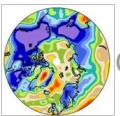


icritc

- The critical mass for auto-conversion controls when ice is converted to snow (precipitation)

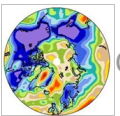
$$PSAUT = C_{i,aut} H(\hat{q}_i - q_{ic})$$

- $.icritc = q_{ic}$
- Where H is the Heavyside function that is 1 when it is positive, 0 when negative and $C_{i,aut}$ is a constant rate.
- See CAM3.0 Description document (eq 4.150) for more info



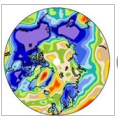
Software Interface Standards: Requirements

- Must conserve vertical integrals of:
 - Mass of each constituents
 - Momentum
 - Total energy
 - Dry static energy
- Must not modify state directly
- Must produce tendencies to modify state



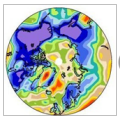
Interface Standards

- Detailed documentation exists on the Physics driver (register, init, tend) and physics state structures
- Utilities are available:
 - Physical constants (shr_const_mod → physconst)
 - Output (already covered this)
 - Physics buffer: place to put variables not in state needed from other parameterizations, or across time-steps
 - Tools for managing constituents and time also exist



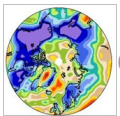
Physical constants

- Physical constants are put in one place in ccsm:
`CAM_ROOT/models/csm_share/shr/shr_const_mod.F90`
- These get remapped in CAM in
`CAM_ROOT/models/atm/cam/src/physics/cam/physconst.F90`
- As an exercise, we will go find some



Final Word: Talk to us

- If you are going to undertake a major piece of code: talk to us (AMP group) first!
- We can help. It may fit with other priorities.
- Good coding and a good foundation makes it easier to 'port' (move) to different versions.
- We may have development versions of the code that would be better than release versions.

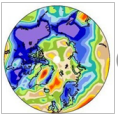


Where to get help

- CAM Documentation (on the web)
<http://www.cesm.ucar.edu/models/atm-cam/>
- CAM Code itself

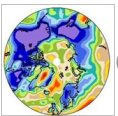
```
$CAM_ROOT/models/atm/cam/bld/namelist_files/  
  namelist_definition.xml
```

```
Configure -h
```
- Bulletin Board:
<http://bb.cgd.ucar.edu>
(good for getting things running on a cluster)



Exercises

- Find things in the code: `grepccm`
 - ‘aqua_planet’, Physical Constants
- Adding an output field (CAM)
- Modifying a parameter in the code (SCAM)
- Impact on model simulations (SCAM, CAM)
 - Visualize, Diagnostics
- Advanced: make a variable and output it (CAM)
- Coding standards, Tracking & Backup

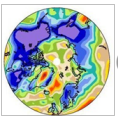


grepccm: how to find things

grepccm is a modified version of `grep` that from a build directory (“bld”) will search for a string in all CAM source code directories

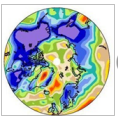
grepccm keys off of ‘Filepath’: a file created when the model is built (compiled)

Let’s try it.



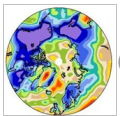
grepccm

- On bluefire: copy the tool into your path
`mkdir bin`
`cp /blhome/andrew/bin/grepccm bin`
- Now go to a model bld directory (mine or yours)
`cd /ptmp/andrew/test/bld`
- Look at “Filepath”
`less filepath`
- Run grepccm
`grepccm aqua_planet`
- Where are the switches for SST fields?



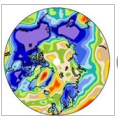
Physical Constants

- Using `grepccm` (from a bld directory):
- What is the value used for the Stefan-Boltzmann constant for blackbody radiation (σ) used in $S=\sigma T^4$
- Where is this found in CAM, CCSM?
- Take a look at these routines to see what is there: other constants
- Questions:
 - What is the 15th digit after the decimal place in π ?
 - What would you change for a CAM-Venus model?



Simple Code Modifications

- One common thing is to output a variable that is not already output from the model
- Example: If you look in the documentation, there are fields for in-cloud water path: ICLDIWP (ice) and ICLDTWP (liquid + ice)
- There is no field for ICLDLWP
- Lets make one

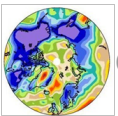


Find ICLDIWP

- Use grepccm from a bld directory on bluefire

```
cd /ptmp/andrew/test/bld/  
grepccm ICLDIWP
```

- Where should we go?



Let's start looking and modifying

- Make your own version of the routine

```
cd ~/tutorial
```

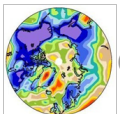
```
mkdir /blhome/$USER/tutorial/mods_icldlwp
```

- Find the full path to the source code specified by a script (`CAM_ROOT`), use it to find the code:

```
cd $CAM_ROOT/models/atm/cam/src/physics/cam
```

```
cp param_cldoptics.F90 ~/tutorial/mods_icldlwp
```

- Let's start looking and editing
 - Search for ICLDIWP as a model



What makes an output variable?

- Note that 'ICLDIWP' only appears 3 times:

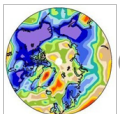
```
call addfld ('ICLDIWP', 'gram/m2', pver,'A','In-cloud ice water path', &
  phys_decomp, sampling_seq='rad_lwsw')
call add_default ('ICLDIWP', 1, ' ')
call outfld('ICLDIWP' ,cicewp , pcols,lchnk)
```

addfld: assigns the variable (in '_init' routine)

add_default: adds it by default to h0 (in '_init')

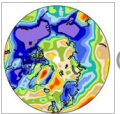
outfld: tells the code to write a variable (cwp) to it (in '_tend')

- Not going to make it default. Just need:
 - addfld (init)
 - outfld (tend)
 - variable to put into outfld



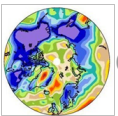
Modify code

- Copy what is there:
 - `addfld` for ICLDLWP that mirrors ICLDLWP
 - `outfld`: what variable is used for liquid?



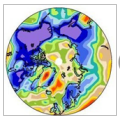
Modify code

- ADD: Copy what is there:
 - Copy the `addfld` line for ICLDIWP and change it to ICLDLWP (also the long name!)
- `outfld`: now copy it for ICLDIWP
 - what variable is used for liquid?
`cliqwp`
 - Copy the `outfld` call and replace `cicewp` with `cliqwp`
 - Replace ICLDIWP with ICLDLWP
- Remember to ‘mark’ your additions somehow in the code



Now Run It

- Modify run script on bluefire
- Copy and make a new script.
 - Note: case name must match `mods_icldlwp`
 - e.g. `case = icldlwp`
- Make sure to add 'ICLDLWP' to namelist in `finc11` (it is not default). Might want ICLDTWP, ICLDIWP as well
- Run the script for 3 months



Did you follow good coding standards?

(separate script, mods directories?)

Did your write down what you did?

BACK UP THIS INFORMATION!

It is usually small (scripts and source code)

Now.

I mean it.

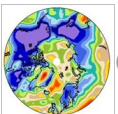
Copy your scripts to your local machine.

```
cd ~/
```

```
tar cvf scripts.tar tutorial/
```

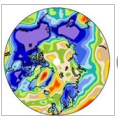
```
FTP.
```

We can wait. The model is running.



Visualize our new Output

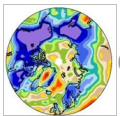
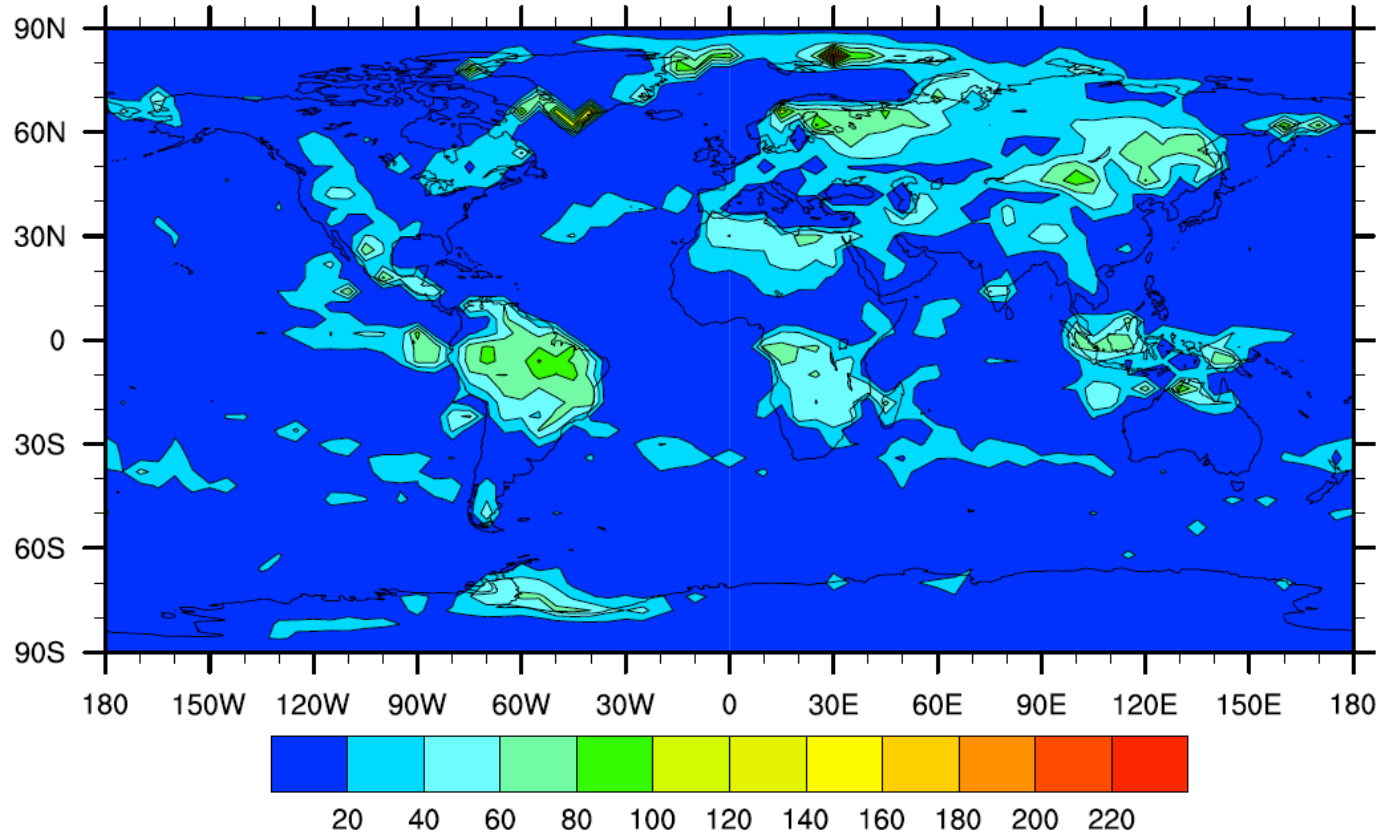
- Let's use the NCL script `atm_latlon.ncl` on storm to visualize ICLDLWP at different levels
 - Modify the case name and field name in the script
 - Look at different levels
- Compare to ICLDTWP and ICLDIWP as well



New ICLDLWP

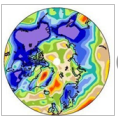
case: test_icldlwp, file: 0000-02

In-cloud liquid water path, day 59, 992.556518e-08 max 229.353 gram/m²



Parameter Adjustment

- Now let's change a number to see what it does.
- Not a physical constant (gravity, pi, etc)
- Could change solar constant! (we just did)
 - That is another story
- Let's pick something unconstrained:
 - Critical mass for auto-conversion of ice to snow:
`icritc`

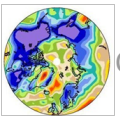


icritc

- The critical mass for auto-conversion controls when ice is converted to snow

$$PSAUT = C_{i,aut} H(\hat{q}_i - q_{ic})$$

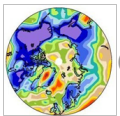
- Where H is the Heavyside function that is 1 when it is positive, 0 when negative and $C_{i,aut}$ is a constant rate. $icritc = q_{ic}$
- See CAM3.0 Description document (eq 4.150) for more info



Find the variable: `icritc`

- Use `grepccm`
from a build directory: `grepccm icritc`
 - Why so many? Look at code.
- Variable is set to different values for different configurations
 - This itself indicates it is not constrained!
- Copy over `cldwat.F90` routine to a new mods directory

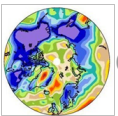
```
mkdir ~/tutorial/mods_scam_icritc
Cp [DIR*]/cldwat.F90 ~/mods_scam_icritc
```
- Lets change it: value is 9.5ppm (9.5e-6)
 - Higher or lower. You decide.
 - What do you think it will do?



Where do I change it?

- What case are we running?
 - FV 4x5
 - Or SCAM: 'eul'
- Be careful! same code will have different values in SCAM (dy core = eul) and CAM (dy core = FV)
- Cheat: Do it at the end of the block
- Can just copy and paste a line:

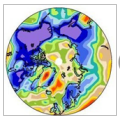
```
icritc = 50.e-6_r8
```



Check your work: output variables

- What values are used for 4x5 FV? For SCAM?
- How do you know?
- You can print the values, or the code does!
- Look at cldwat.F90 again and search for 'icritc'
 - L347: `write(iulog,*)'tuning parameters cldwat:
icritw',icritw,'icritc',icritc,'conke',conke`
- Now look in an output file:

```
cd ~/tutorial  
grep 'tuning parameters cldwat' out.*
```

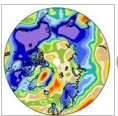


A few more notes

- ‘masterproc’ tells it to only do the commands on startup.
- You can output anything to the log file
- Also: you might want to rename the log file in the script (at the top)
 - Now it has job number. You could manually make it the case name in the script. Just change:

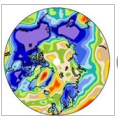
```
#BSUB -o out.%J
```

```
# output filename
```



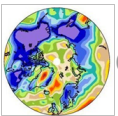
Notes: Coding Standards

- What is with the ‘_r8’?
- Specifies to the code that the value is to be exact for a 32 bit (4 byte, double precision)
- It adds a lot of zeros in the computer, otherwise you can end up with 40.000000032 or something
- This is a compatibility issue with fortran code – to insure that the precision is controlled



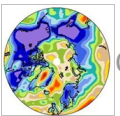
Run the new code: SCAM

- Copy the SCAM script to a new case name
e.g: `scam_icritc50ppm.csh`
- Modify code with the right case name
`scam_icritc50ppm`
- Code has to find `mods_[case]` directory
- Run it in SCAM: use the standard arm95 case



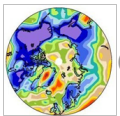
Visualize in SCAM

- Once the scam run completes, use NCL to visualize it on storm with `scam_latht.ncl`
`/fs/home/andrew/ncl/scam_latht.ncl`
- Is it different than what you have run before (`scam_test` or `scam_test01`)?
- How can you really tell?
 - What variable might show differences?
 - Can you change contour intervals?
 - What about a Difference plot (next slide)!



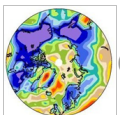
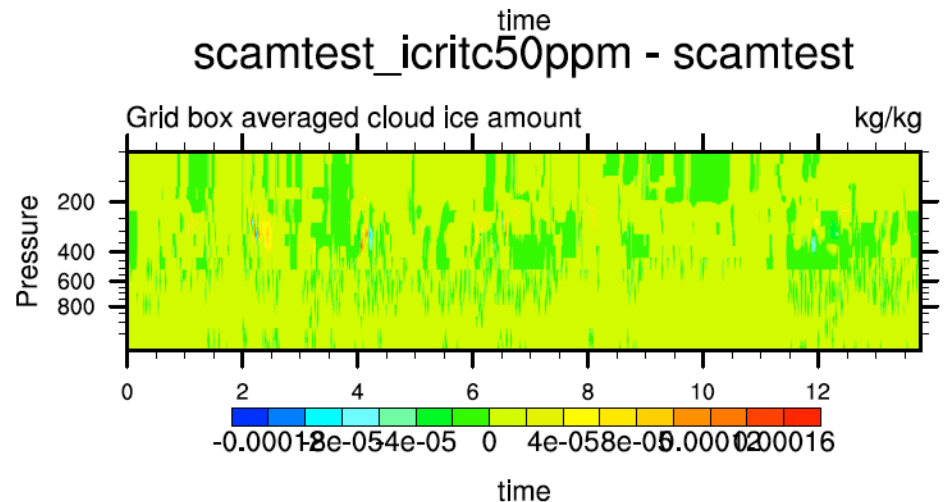
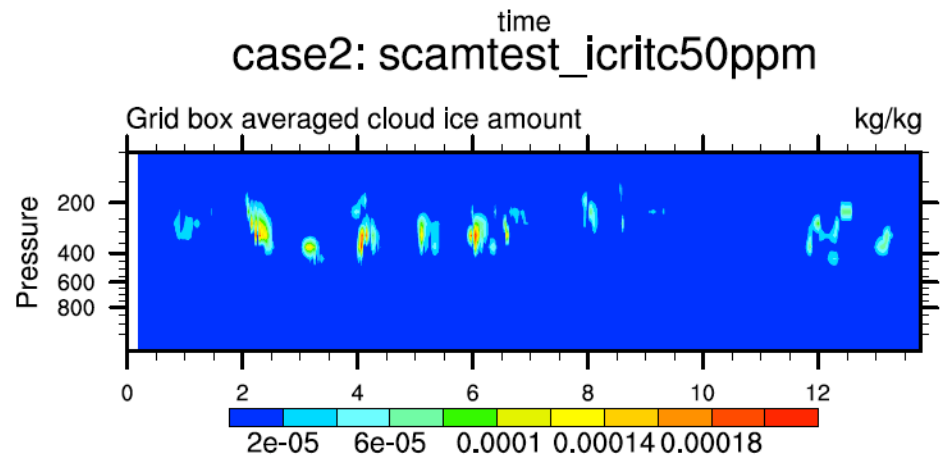
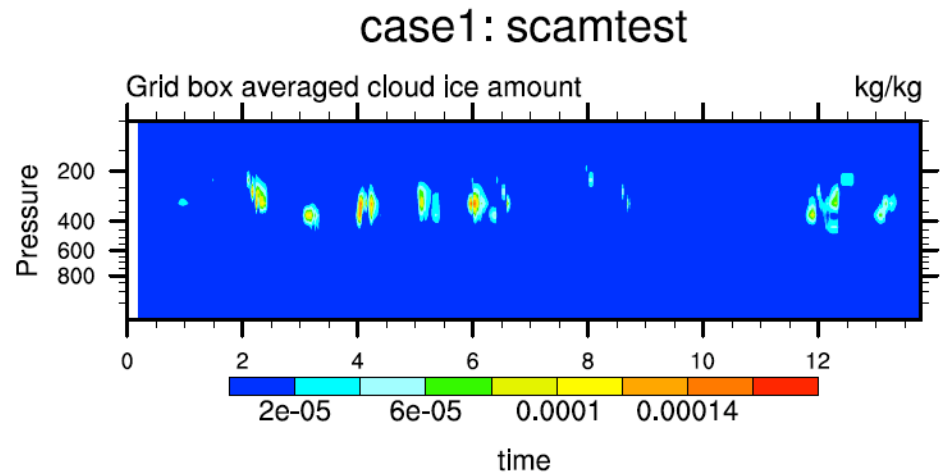
Visualize differences in SCAM

- Let's compare two cases: the basic SCAM case and the one we just ran.
- Do this with another NCL script (on storm):
`/fs/home/andrew/ncl/scam_diff_latht.ncl`
- Copy the script
- Set paths, and now set 2 case names
- Run the script...
- Look at the results



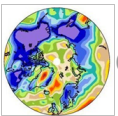
SCAM: Differences

- More or less ice?
- Explore: Change contour intervals for the difference plot!
- Look at sample in script. Can you figure out how to do it?



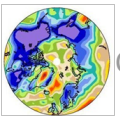
Run in the full model

- Now let's take the changes and make a new case to run CAM
- Back to bluefire.
- Pick a case name and copy a run script for it
- Copy the mods_ directory to the new name
- Run for 3 months



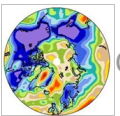
While that is running: Diagnostics

- I already ran this case for 2 years:
`/ptmp/andrew/test_icritc50ppm`
- Let's go over to storm and run the diagnostics on the difference!



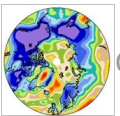
Run Diagnostics for 2 models

- Point diagnostic code to my directories on bluefire (test_path, cntl_path)
- Run and copy diagnostics over to a local machine (see practical 2, part 2)
- What do you see?
- Can you explain the effects?
 - What makes sense, what did you not expect?



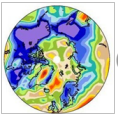
Advanced: Make a Variable

- How about the minimum layer water vapor in a column?
- Is this diagnostic or prognostic?
- How would you write it in fortran?
 - generate some pseudo code (probably a loop over i columns)
- Where in the code would you put it?
- Think about how water is modified. Where in the physics?



Minimum Water Vapor

- Probably in the driver for the stratiform clouds
 - This is the last condensation package run
 - Let's try `stratiform.F90`
 - Also, notice lots of output
- Steps
 - Addfld call in `stratiform_init`
 - Note: find a single level field to copy! (e.g.: `CDNUMC`)
 - Name and units (mass mixing ratio kg/kg)
 - Add an array to `stratiform_tend` (local variable)
 - Near bottom of `stratiform_tend` fill the array
- Remember to mark all your code changes



Minimum water vapor

- When filling the array
 - Initialize first. (set to zero)
 - Loop over columns ('i' dimension)
 - Again, other code can be used as a model
- After: output the new variable
 - `outfld` call with output field and variable field
- Compile and run: remember to add the output variable to the namelist in `fincl1` !
- Visualize as before

