-

,
-
,
. , -
, -
, ,
, .

, 0 0

! "

##
GENERAL SOFTWARE AND RAMS INSTALLATION INSTRUCTIONS
##

RAMS IS DESIGNED TO RUN ON A LINUX x86_64 PLATFORM. IF YOU HAVE A DIFFERENT PLATFORM, CERTAIN SECTIONS OF THE CODE WILL NEED TO BE CUSTOMIZED BEFORE THE MODEL CAN BE COMPILED SUCCESSFULLY AND USED.

RAMS IS COMPOSED OF C AND FORTRAN-90 FORMAT CODE. YOU MUST HAVE C AND FORTRAN COMPILERS INSTALLED ON YOUR SYSTEM BEFORE PROCEEDING. "GCC" IS THE MOST COMMONLY USED C COMPILER WITH RAMS BUT HAS ALSO WORKED WITH "CC", "MPICC", AND "ICC" ON SOME SUPERCOMPUTERS. WE HAVE SUCCESSFULLY TESTED RAMS WITH THE FORTRAN COMPILERS "PGF90", "IFORT", AND "GFORTRAN". IT HAS COMPILED WITH "MPIF90" ON SOME SUPERCOMPUTERS. OTHER COMPILERS MAY NOT WORK WITHOUT CODE MODIFICATIONS. NOTE THAT RAMS COMPILED WITH "GFORTRAN" RUNS MUCH SLOWER THAN OTHER COMPILERS AND THE CODE CAN BE PROBLEMATIC AT RUNTIME. GFORTRAN SEEMS TO FLAG ERRORS THAT ARE OFTEN NOT REALLY ERRORS SUCH AS "ARRAY BOUNDS" THAT ARE NOT ACTUALLY A PROBLEM.

Outline:
Directory description and overview

##

##
1. First unzip (gunzip <filename>) and untar (tar –xf <filename>) the RAMS release.

2. When you unpack the model release it will open into two main directories. (1)The RAMS main directory named like: “rams_20170906_release_6.2.06”. (2)The geographic surface data directory (needed for real case studies) named: “sfctypehdf5”. It is typical to place these directories side by side in a location such as “/home/username”. The location of “sfctypehdf5” has to ultimately be specified in the “RAMSIN” namelist control file that is described below.

3. Enter the main RAMS directory such as “rams_20170906_release_6.2.06”. This top-level directory path will be the "RAMS_ROOT" environmental variable used in subsequent steps in the “include.mk” compile control file. Modify the "include.mk" compile control file with correct paths to the RAMS source code, RAMS version number, linking libraries, HDF5, MPICH, Fortran compiler and flags, C compiler and flags, and MPI flag section. Get to know your "include.mk" file! Changes will need to be made for different computers in order to get the model to compile. For an initial serial compilation (running on only 1 processor), comment out the "PAR_" flags at the bottom of "include.mk".

4. Once your "include.mk" file is correct for your computer system, enter "bin.rams" directory and type "make clean" to clean up an old compilation, and then type "make" to compile the model. Any errors that show up are likely due to issues with settings in "include.mk". You will need to fix any such discrepancies.

5. Once the model is compiled, it will produce an executable like "rams-6.2.12".

Below is a description of the files and directories you see in this top-level location. We will step through the files and directories in an order that will allow you to compile and test the various components of the model as you read through this document.

##

-

##
This DIRECTORY contains HDF5, MPICH, SZIP, ZLIB third-party software. RAMS needs the executables and/or libraries and include files associated with these packages. HDF5 needs the SZIP and/or ZLIB libraries. Your user environment specified in Linux files such as “.bash_profile”, likely needs to point to these “bin”, “lib”, and “include” directories in the “PATH=” environment variable. The “LD_LIBRARY_PATH=” likely also needs to include the paths to the libraries for each of the software packages. A lot of this depends on how your Linux system and user environment are set up. If you get compile errors with RAMS, pay attention to the error messages, as they often let you know if certain executables, include files, and libraries are not accessible. If they are not the error usually lies in the linking in “include.mk” or your system environment variables. Once you have a compiled RAMS executable such as “rams-6.2.16” you can run “ld rams-6.2.16” to see which libraries are linked to make sure you are using the expected versions in the event that you have multiple versions on your Linux system.

First, we start here assuming that your computer system has the PGI (pgf90) or INTEL (ifort) Fortran compilers and a C compiler. RAMS may or may not work with GFORTRAN.

RAMS will not compile without these third-party software packages. Attention to this must come first before proceeding. If this software is already on your computer system or supercomputer that you are using, then you do not need to install these. Just make sure your Linux system includes the "PATH" and/or "LD_LIBRARY_PATH" to these software packages. and that your "include.mk" compile file points to the necessary software.

The included software is needed by RAMS for parallel input/output (HDF5) and parallel processing (MPICH). RAMS input/output is done with parallel-HDF5 for use on massively parallel systems. The included third-party software are versions that we use frequently and are well tested with RAMS. You can install other versions if you wish. However, you must have some version of parallel-enabled HDF5 installed on your computer. Parallel-enabled HDF5 requires Message Passing Interface (MPI) software installed on your computer. This "misc" directory contains pre-compiled (for Linux x64-86 system) versions of HDF5, MPICH, SZIP, and ZLIB. Use the default pre-compiled versions first to see if they work. You can place pre-compiled directories somewhere on your computer system and then make sure your "include.mk" compile control file points to the paths for this software. Also make sure your Linux ".bash_profile" type environment file sources the locations of HDF5, MPICH, SZIP, and ZLIB executables/libraries in the "PATH" and "LD_LIBRARY_PATH" environment variables.

More information on this topic is given in the README file in the "misc" directory where the software lives.

RAMS and third-party software need C libraries, so glibc and a C compiler need to be available on your computer system and they usually are by default on Linux systems. As you move RAMS between systems or have Linux OS upgrades, you might need to recompile third-party software and RAMS due to changes in glibc and other system libraries.

See section on "include.mk" for more details on use of these necessary software packages.

##

. .

##

This FILE is the compile control file.

This is where the user sets environmental and directory paths for RAMS source code, HDF5, MPI, and FORTRAN and C compilers as well as the compiler flags and required libraries. It is the only file that needs to be modified prior to compiling within the various RAMS top level "bin" directories. This file requires correct setup before any compilation will work.

Edit the include.mk file. Several things in this file need to be changed/set:

a. Change the RAMS_ROOT variable to point to the top-level directory and change the RAMS_VERSION number if necessary.

b. HDF5 directory location – HDF5_ROOT. Set HDF5_LIBS to the correct library paths of your HDF5 installation. Set HDF5_INCS to the directory where the C include file 'hdf5.h' exists. Leave HDF5_DEFS alone for now. Try the default HDF5 settings first that point to our pre-compiled HDF5 libraries. In you are running on a supercomputer, try our HDF5 or the native computer versions installed as modules. Commands on supercomputers such as "module avail" or "module list" can provide info on what software is available and how to load it.

c. MPI directory location – MPI_ROOT. Set PAR_INCS to the directory where the MPI include file exists. Set PAR_LIBS to the correct library paths of your MPI installation. Set PAR_DEFS = -DRAMS_MPI. Try the default MPI settings first that point to our pre-compiled MPICH2 libraries. If you are running on a supercomputer, it is best to use their MPICH modules! See their instructions on how to load modules.

d. Machine-dependent options:
CMACH - name of machine/compiler type. Used in source files to conditionally compile code sections. Our include file and code currently only contains 'PC_LINUX1' option for Linux framework. If you have a very different platform, do a code search for 'PC_LINUX1' to see where you need to make modifications. These are related to specific command line argument collection, clock timing methods, and use or non-use of C subroutine underscores. You may need to examine these files and modify code for your system since PC_LINUX1 is our currently tested platform.

e. Fortran and C compiler flags:
There are default Fortran compiler flags already included for PGF90, IFORT, and GFORTRAN; uncomment only one set of these and insert the proper path to compiler. The Fortran flags need to either reflect the single or double precision variable option. There are default settings for both in this file. These settings are initially set to single precision. Note that double precision will reduce runtime, use more memory, and increase the output file size substantially. The default C compiler is GCC.
F_COMP - name of Fortran compiler.
F_OPTS1 - command line options for the Fortran compiler.
F_OPTS2 – command line options for the Fortran compiler.
C_COMP - name of C compiler.
C_OPTS - command line options for the C compiler.
LOADER_OPTS - options for linking.
LIBS - any other libraries required for linking.
ARCHIVE - archive command. Typically 'ar rs'.

F_OPTS1 and F_OPTS2 can be the same or different. We have found that we need to compile some Fortran files with different optimization levels; thus we typically have these two flags differ in their optimization level only. In the RAMS MAKEFILE you can see which files are compiled differently from others.

( ) , , , ,
, ( ) . , ( )
(.
.), .
.

##
. .
##
This DIRECTORY is the primary RAMS model compile directory.

Enter the bin.rams directory. Perform a fresh compile by typing "make clean" and then "make". Here you will create an executable named something like “rams-6.2.06”. If you get a compile error please see section on troubleshooting.

The executable is controlled by the “RAMSIN” namelist control file which is used to specify your simulation setup and choices of parameterizations. A detailed description of each namelist variable is given in the docs file RAMS-Namelist.pdf. Simple sequential (single-processor) RAMS model execution is done with the syntax example “rams-6.2.06 –f RAMSIN”. You can change the name of the RAMSIN namelist as long as you specify the namelist being accessed. Parallel execution varies with MPI software. A sample execution using MPICH2 and machines file “machs” would be something like:
“mpiexe –machinefile machs –np 12 rams-6.2.06 –f RAMSIN”.

In general, there are 2 main types of simulations that can be run with RAMS: (1) idealized, and (2) case studies. Idealized simulations do not require the geographic surface data in “sfctypehdf5” or any data pre-processing of gridded reanalysis products since they are typically initialized from a sounding and specified surface characteristics in the RAMSIN namelist and do not use boundary nudging. Variable initialization case study simulations that are tied to specific and real geographic locations require that you point to the location of “sfctypehdf5” in RAMSIN. Case studies also require that you provide gridded reanalysis type data in grib-1 or grib-2 format that is formatted into “dp” files via the data preprocessors in “bin.dp.grib1” or “bin.dp.grib2”, respectively. The “dp” files created in the bin.dp.grib directories is pointed to in the RAMSIN in order to generate the varfiles for variable initialization. The RAMSIN namelist guide provides all the details of the settings.

There are 2 test simulations here:

1. Idealized supercell thunderstorm test on small domain using a sounding for horizontally homogeneous initialization (Leah Grant et al. 2014,2015, JAS). The script "run_test_supercell.sc" can be used to execute this test. Standard runtime output (not model analysis files) is sent to the screen for viewing progress of the simulation. It is a very fast running simulation used to see that your installation is working for a basic simulation. The script uses the RAMSIN namelist file "RAMSIN.supercell". After the simulation completes via the execution script, it also executes the REVU post-processor (if REVU has been compiled in “bin.revu”) for several model data fields (see section bin.revu for information about REVU). It uses the REVU

namelist "REVUIN.supercell". This REVU namelist is setup to output in TEXT format for quickly viewing that data is present in the output files. The output from this simulation is sent to directory "test.storm" which is created automatically in the execution script.

2. Winter orographic case study simulation for event that occurred over the Park Range of Colorado on Feb 11, 2007 (Stephen M. Saleeby et al. 2009, JAMC). This case study simulation is executed by the script "run_test_orographic.sc" and uses the namelists "RAMSIN.orographic1" and "RAMSIN.orographic2". The first RAMSIN file is used to create the surface files and varfiles for this case study since real surface data and real meteorological data is required to run a real case event. The "MAKEVFILE" runtype set in the namelist RAMSIN.orographic1 specifies that simulation surface files are generated from the data in "sfctypehdf5" that was packaged within the model release tarball. After generating the surface files that are specific to this simulation, it proceeds to create the variable initialization files (or "varfiles") from our pre-created dataprep files (or "dp" files). The "dp" files for this case sit in the directory “dprep.test”. These were created using NCEP/NCAR 2.5deg global gridded reanalysis and the data preprocessor compiled within “bin.dp.grib2”. The NCEP/NCAR reanalysis data are not provided for this test. The output from this simulation is sent to directory "test.orographic" which is created automatically in the execution script. There is no REVU post-processing portion for this test simulation since this was already tested in the idealized supercell test discussed above. The second orographic test RAMSIN file mentioned above is then used for running the actual orographic case study simulation.

3. Note that both test scripts, run_test_supercell.sc and run_test_orographic.sc, are set up to run in sequential mode and not use parallel processing. They will run faster if you enable parallel processing. Please examine these scripts to understand their usage and how to turn on/off the parallel processing. If you turn on parallel processing, via the script flags "runtype" and "n" (for number of processors), the script will attempt to create a machines files (or "machs" file) based on the hostname of the current machine. If your current machine is not multi-core or does not contain the minimum number of cores specified by the script variable "n", then the default parallel test simulation setup will not run correctly without customization of a "machs" file that is appropriate for your computer and MPI software. This is why the default scripts for executing these test simulations are setup to run sequentially and not in parallel. A sample “machs” file named “machs.master” is provided in “bin.rams” and contains several commented out sections that could be used in a similar manner on a local cluster if you have on. Each cluster is different, so the “machs” file would need to be customized for your machine. Such a “machs” file is not needed if running on a supercomputer that uses a “PBS” queueing system. You would need to write your own script to run their “PBS” system with a “qsub” command or something similar.

##

. . -

##

This DIRECTORY is the primary REVU post-processor compile directory.

Enter the bin.revu directory. Perform a fresh compile by typing "make clean" and then "make". Here you will create an executable named something like “revu-6.2.06”. Please see the section on troubleshooting if your compile does not work.

The executable is controlled by the "REVUIN" namelist control file. REVU execution is done with the syntax example "revu-6.2.06 –f REVUIN". You can change the name of the REVUIN namelist as long as you specify the namelist being accessed. Information on the variables in the REVUIN namelist are given in the file REVUIN.example.

REVU is a post-processor that will read the HDF5 format output files from RAMS and output any of a number of variables listed in the docs file RAMS-OutputVariablesREVU.pdf. Some variables in this list are native RAMS predicted variables and other are post-computed derived quantities. Alternatively, many plotting packages such as Matlab, IDL, Python, etc., can read the native HDF5 RAMS model output. A complete list of output native RAMS variables is contained in the docs file "RAMS-VariableList.pdf".

Examine the REVUIN example to see how to modify it for reading a given simulation's output data and writing out the chosen variables. You can write output from REVU in HDF5 or TEXT formats.

Output data files in HDF5 format are named like:
"a-AS-1991-04-26-210000-g1.h5".

Output data files in TEXT format that can be used to create GEMPAK files or files for the Unidata-IDV are named like:
"a-AC-1991-04-26-210000-g1.txt".
The associated ".tag" file gives some GEMPAK/IDV grid specifications.

Within the file name convention:
The "g1" indicates output on grid-1.
The "a-AC" indicates output on Cartesian grid.
The "a-AS" indicates output on sigma-Z grid.
The "a-AP" indicates output on pressure levels.
The "a-AG" indicates output on soil/below-ground levels.

Data in example_data subdirectories z.test.mc3e3, z.test.spl1, z.test.storm5, z.test.storm8 were created from RAMS version 6.1.22 but are still valid for testing up through this latest version.

In the REVUIN namelist file you can choose to output fields of your choice at chosen times and levels. The current REVU output format options are TEXT and HDF5. The HDF5 output files can be viewed with appropriate software including GRADS (sdfopen), MATLAB, IDL, PYTHON, etc. The TEXT output files can be used to create GEMPAK gridded files that can be viewed in GEMPAK software and the Unidata IDV (Integrated Data Viewer).

Note again that REVU is not necessary for examining and plotting model output but it can be useful for obtaining many derived quantities such as "vertically integrated condensate" that would otherwise be more difficult to compute. See the separate REVU document for details.

##

. . . - -

##

This DIRECTORY is the compile directory for the RAMS GRIB-1 pre-processor.

Enter the bin.dp.grib1 directory. Perform a fresh compile by executing the "make.sc" bash script. Here you will create an executable named something like “dgrib-6.2.06”.

The make.sc script must first compile “rams_wgrib.c” and tries to use the default gcc compiler. If gcc cannot compile this “.c” file, then you will need to modify make.sc and try a C compiler on your computer that works.

What the de-gribber does

This de-gribbing software is used to extract and convert Grib-1 formatted gridded atmospheric pressure level data into RALPH-2 ASCII-format dataprep or "dp" files that RAMS uses for making variable initialization files or "varfiles" used for real case study initialization and nudging. The "dp" files have header information at the top followed by the data that is ordered in the file according to the order of the variables listed in the standard output that is sent to the screen when running the de-gribbing executable.

These “dp” data are needed for case studies in which heterogeneous initialization and lateral boundary or central data nudging are necessary. These are not needed for idealized type simulations that use homogeneous initialization from soundings and no nudging. More specific information on preparing gridded atmospheric data, observed rawinsonde data, or observed surface data can be found in the docs file RAMS-DataPrep.pdf. Note that for real case study simulations, only processing of gridded datasets is required; rawinsonde and surface observations are not necessary, but they can be input as a customization used by the model in the observational data assimilation (ODA) section noted in the RAMSIN namelist.

What the de-gribber extracts

Atmospheric fields:

The degribber code always tries to extract the atmospheric fields of u-wind, v-wind, relative humidity or specific humidity, temperature, and geo-potential height. It assumes winds are earth relative. There are some customized exceptions that are handled on a case by case basis and are mentioned below. If a dataset does not contain these default conventions, then customization is required in the dataprep source code in dprep/dgrib1_main.f90 and in lib/griber_grb1.c for Grib-1 format.

Surface fields.

The code also tries to extract soil moisture, soil temperature, snow depth, and snow water equivalent when provided.

Executing the de-gribber

! *

Run as "dgrib-<version> -t # -d YYYYMMDDHH -h fcsthr -f filename"
1. -t specifies the data set being used (number as above)
2. -d is the date and time
3. -h is the forecast hour (usually zero for reanalysis
4. -f is the path to the file to convert
Where the -t argument could be “1” for NCEP-reanalysis.
Where the -d argument could be “2007021100” for 00Z Feb 2, 2007.
Where the -h argument could be “0” for forecast hr 0 (almost always 0).
Where the -f argument could be “/home/smsaleeb/NARR_20070211.grb”.

- .
.

/ . **0** / .
- .

##
. . . - -
##
This DIRECTORY is the same as bin.dp.grib1 but for working with gridded data in GRIB-2 format.

Enter the bin.dp.grib2 directory. Perform a fresh compile by executing "make clean" and then "make".

For this de-gribbing code to run:
This software requires that the wgrib2 software be correctly compiled on your machine. Here we have provided a precompiled executable "wgrib2" and the wgrib2 software source code wgrib2.v2.0.3.tar.gz. If the precompiled executable does not work for you, then you will need to try to compile the code in wgrib2.v2.0.3.tar on your computer and place the generated executable "wgrib2" within this current directory in place of the provided precompiled executable. To compile wgrib2 from source, unzip and untar the wgrib2 source code provided; enter the main directory. Read the INSTALL file and “makefile” to be a bit familiar with them. You might need to make a few changes. However, you can first try typing “make” to see if the installation settings work as is. If so, then you will find a “wgrib2” executable within the “wgrib2” directory. You can copy this executable to the bin.dp.grib2 directory and try out the example tests as discussed below.

This code for Grib-2 format accomplishes the same task that the section above for Grib-1 format. It is just that it uses the Grib-2 de-gribber uses the code dprep/dgrib2_main.f90 and lib/griber_grb2.c which are similar but account for some differences between Grib-1 and Grib-2. The big difference with Grib-2 is that the third-party “wgrib2” software is a bit more complex to install than “wgrib1”.

! "+

- - .
, .
. 0 . 0.
.
.
##
. . - /
##

- . (.)

Enter directory bin.block. Perform a fresh compile by typing "make clean" and then "make".

All of the static surface characteristics data in “sfctypehdf5” (ie. vegetation, soil type, topography, SST) are in block file format, meaning that multiple files are used to cover certain latitude longitude blocks of the earth rather than having a single very large file that covers the entire earth. The RAMS model is programmed to read this specific file format for surface characteristics. The example setup in this directory is for creating SST files from observational data. To get info on how to execute, just run the executable after compilation, and usage information will be displayed to the screen. (This requires in-depth model knowledge and customization for any usage.)

To test BLOCK file making, you can use sample MODIS-4km and REYNOLDS-1-deg SST data in "example_modis_sst" and "example_reynolds_sst" that has been pre-prepared in a format that the blockfile making code can ingest (These are not provided in the standard model release but are available upon request). Just modify and run the script run_test_sstfiles.sc. This will output new SST block files that are in a format that RAMS can ingest for making runtime surface files in a MAKESFC run. The scripts will automatically compare the newly generated block files to some pre-existing ones that are in the example directories.

To run the block file executable:
executable input_file datatype output_directory prefix

Whereby:
input_file - name of input data file
datatype - 1=modis9kmsst, 2=1degReynoldssst
output_directory - directory where header file will be written
prefix - filename prefix for header and data files

Example execution with sample file names and such:
mk-blkfiles-6.1.0 131231_ 1 output S

! " "

See SST example in directory example_modis_sst.
File 131231_ is ASCII lat,lon,SST(C) MODIS data.
Files S131231_90S000E.h5 S131231_90S180W.h5 are output for RAMS SST.
File SHEADER is the header file.

.

" "

" . 0".

##

.

##
This DIRECTORY contains the files of RAMS documentation.

There is ample documentation regarding many aspects of the RAMS model. Several key documents to note in order to run the analyze model results are as follows.

1. RAMS-Namelist – This describes all of the key parameters that need to be set in the RAMSIN namelist file in order to start a RAMS simulation.

2. RAMS-VariableList – This contains a list of all possible RAMS output variables that could be contained in the output analysis files. It contains the name, description, and variable units.

3. RAMS-OutputVariablesREVU – This contain a full list of all possible variables to output via the REVU post-processing package. It provides the name and units of the variables that can be extracted or generated from RAMS output. The variable names found here are those that are to be used in the REVUIN namelist file.

4. RAMS-Updates – Provides a detailed log of changes from one version of the model to the next.

5. RAMS-DataPrep – Provides a detailed description of the formatting of the dataprep or “dp” files that are generated from gridded Grib format datasets. These “dp” files are used by RAMS to create the variable initialization and nudging files or “varfiles”. The varfiles are then used to initialize and/or nudge the model in case study simulations.

6. RAMS-TechnicalManual – Provides a summary of the parameterizations and mathematical techniques used in RAMS.

7. RAMS-Leaf2-to-Leaf3 – This provides some information on the changes in the LEAF land-surface model from versions 2 to 3.

8. RAMS-HUCM-Bin-Micro – This provides information on how to run RAMS with the Hebrew University Cloud Model with Spectral Bin Microphysics. It also provides information on the output variable names.

##

•

##

This DIRECTORY contains the directory tree to RAMS source code.

##

0.

##

This DIRECTORY contains files that the model accesses for information related to particular subroutines.

Note that the location of this directory and files will need to be input into the RAMSIN namelist at runtime.

##

##

1. TO PROCESS OLD ANALYSIS FILES WITH NEWER REVU:
We used array swap starting in RAMS 6.1.12. To process old data with new REVU code, look in the file "lib/hdf5_utils.f90" for notes by "Saleeby" about a few lines of code to change. Do not forget to change this back after working with old data.

2. TO PROCESS RADAR PARAMETERS SUCH AS D0, LOGNW, ETC. FROM REVU WITH OLD DATA:
Some older simulations do not contain GNU (hydrometeor shape parameter) in the header files, so we need to temporarily hardcode the value of GNU used in the older simulations we want to examine. This can be done in subroutine "hydrogamma" in the file "revu/rcomp.f90". See notes by "Saleeby".

3. CANNOT FIND COMPILER, HDF, OR MPI
Most compile problems and error messages tend to be related to improper installation of

overflowing. To avoid this, you can increase your Linux system stack size or set it to unlimited. There should be no real problem with this. The command to do this is “ulimit –s unlimited”.

6. RANDOM CRASHES DUE TO MAX VARIABLE HARDCODED LIMITS
When you first run the model, note that most memory is dynamically allocated. However, since some variables must be declared before the model has the chance to read the RAMSIN namelist to obtain model dimensions and such, some variables must be given "max" dimensions. The source file "memory/grid_dims.f90" contains a list of "max" dimensions. You should adjust these as needed for your simulations. The dimensions are currently set to fairly large values, but as modeling capabilities improve, sometimes some of the values of max dimensions need to be increased.

7. RUNTIME “TOO MANY CCN” TYPE ERROR:
This error is usually indicative of numerical problems near topography. Try reducing your timestep and/or using RAMSIN variable IHORGRAD=2 to deal with diffusion near steep topography. You can also try smoothing your topography to prevent adjacent grid cells from having too large of a jump in elevation, which creates the error when using IHORGRAD=1.

8. ISSUES THAT OCCUR WHEN RUNNING WITH FINE VERTICAL SPACING OVER TERRAIN:
First try smoothing terrain using RAMSIN namelist settings for topography representation and topography wavelength. Also try IHORGRAD=2 which allows for small DZ.

9. SMALL NUMBER ARITHMETIC:
Some compilers (IFORT) may take arithmetic of (0.0 + 1.e-12) as equal to 0.0 if the exponential is quite small. If this sum is in a denominator, this will result in NaN. There are many places in RAMS that do this to prevent division by zero and some, but not all, of them have been replaced with something like [max (0.0, 1.e-12) = 1.e-12].

10. ARRAY BOUNDS ERRORS:
Note that NaNs can occur due to array bounds errors. So, if a NaN occurs, turn on bounds checking in the compiler flags to isolate the error location and fix the bounds issue.

11. UNINITIALIZED VARIABLES:
Note that NaNs can occur from uninitialized variables. Some compilers just assign a value of zero and others assign NaN. Turn on compiler checking flags for finding uninitialized variables.

12. Error message: UNDEFINED REFERENCE TO…
One cause of this compiler message could be trying to declare a function within a subroutine when both are in the same module. Only declare the function if they are NOT in the same module.

13. Error message: NEGATIVE CONDENSATE MICRO
This error indicates a problem exists but not the source. This can be caused by CFL errors, cumulus parameterization, diffusion, or advection issues. To check further, turn on ICHECKMIC

in the RAMSIN namelist. See if other error messages appear. Most times, using a shorter timestep solves the problem.

14. MPI ERRORS:
MPI routines hardcode the variables “nwords” in order to set the buffer size for memory before passing data to the nodes in parallel. When adding variables to the put/get list, “nwords” has to be incremented. If the buffer is not big enough, the model will crash and give errors such as “error stack”, or “size of data to pack is larger than remaining space on buffer”.

15. INITIALIZATION IN PARALLEL:
Some initialization needs to be done on a single node and then distributed to all the nodes. As an example, the random bubble generator (IBUBBLE=3 in RAMSIN) will create different perturbations if the sub-domain sizes are different for different numbers of nodes being used for parallel processing. We can get around this by letting a single node do the randomizer at model start and then broadcasting the bubble field to the nodes.

16. NON-IMCREMENTING DO LOOPS
Some compilers do not like non-incrementing DO loops such as [DO N=K1,K2] where K1=K2. In some cases, the loop does not get run, so variables inside the loop are not solved or set. This could lead to NaNs and model crashes.

17. COMPILING HDF5:
In compiling HDF5, the variable H5_CFLAGS may contain the flag "-ansi". This argument tells gcc to use the ISO C90 standard which does not accept C++ style comments which HDF5 seems to use sometimes. So, it recognizes the comment style but will not accept it. The fix is to replace "-ansi" with "-std=c99". This may be contained in "config/gnu-flags" in variable H5_CFLAGS= "$H5_CFLAGS $arch –ansi –pedantic". After switching out the "-ansi" flag, rerun "configure" and check inside the Makefile and verify that H5_CFLAGS has "-std=c99" instead of "-ansi". The try running "make" again.

##

, , , ,

##

THE INSTRUCTIONS THAT FOLLOW IN THE REMAINER OF THIS SECTION ARE ONLY NECESSARY FOR COMPILING/INSTALLING FORTRAN COMPILERS, HDF5, MPICH, SZIP, and ZLIB FROM THE SOURCE CODE. IF THESE ARE ALREADY ON YOUR SYSTEM AND THE PRECOMPILED HDF5 AND MPICH BINARIES WORK FOR COMPILING RAMS THEN YOU WILL NOT NEED THESE INSTRUCTIONS.

1. Install you favorite Fortran compiler. RAMS has been tested with PGI, INTEL, and GFORTRAN. However, RAMS runs slower with GFORTRAN and may not work will with certain GFORTRAN versions.

For using an example Intel fortran compiler (installed this first):
ifort12.compxe_ia64_2011.8.273.tar.gz

a. run ‘install.sh’ and follow commands
b. passcode serial number is ‘XXXX-XXXXXXXX’
c. in your .bash_profile, add
PATH=/opt/intel/composer_xe_2011_sp1.8.273/bin/intel64

SEE THE README FILE IN THE “MISC” DIRECTORY ON INSTALLATION OF OTHER THIRD-PARTY SOFTWARE.

##########

•

##########
SEE MY SAMPLE FILE IN THE “MISC” DIRECTORY

##########

##########
Notes on a few specifics if you are contributing code to RAMS. We are starting to force some general coding practice to make doing code searches easier.

Note: all module, subroutine, and function names should be lower case and all calls or usage of them should be lower case. This makes for ease of direct ‘grep’ matching and script searching when examining code usage.

1. Head every module, subroutine, interface, and function with letter case-specific opener. This should begin at column1 and include the spaces and parentheses.

for subroutine example: ‘subroutine <routinename> ()’
for module example: ‘module <modname>‘
for interface example: ‘interface’
for function example: ‘integer function <funcname>‘

2. Use ‘implicit none’ for every program, subroutine, module, and function and place in first column. The does not have to be added for ‘entry’ statements within subroutines.

3. End every program, module, subroutine, interface, and function with letter case specific closure:

‘End Program main’
‘End Subroutine <namehere>‘
‘End Module <namehere>‘
‘End Interface’
‘End Function <namehere>‘

4. Module ‘use’ statements should be in column 1 before ‘implicit none’.

5. For calling subroutines, use ‘CALL’ in all capital letters, place a

space after the name of the subroutine to call, and always use ()
after the name of the subroutine to call.
(ex. ‘CALL cldnuc ()’)

6. When using a function in a routine, make sure to declare the function as an ‘external’ variable and make the function call name all lower case and put parentheses right after the name.
(ex. ‘real, external :: rslf’) (ie. ‘rslf()’)

7. For compiling pre-processor code that uses “#ifdef” statements, the fortran files need to end with .F90 rather than .f90.

8. For using the C interface in fortran code to read data from HDF5 files, you must get the C trailing character for fortran to be happy. Example from Quickbeam read of dimensions: [must include the //char(0)].
Call rh5d_open (fileid, ‘t_coords’//char(0), dsetid, hdferr)
Call rh5a_read_anyscalar (dsetid, ‘units’//char(0), stime, RHDF5_TYPE_STRING, hdferr)

##

##
WGRIB/WGRIB-1(same thing):
To see everything in a Grib-1 archive:
wgrib <gribfile>
To see header and grid info do:
wgrib -V <gribfile>
To extract a certain field using grep and write to a new GRIB-1 file:
wgrib -s <gribfile> | grep ":UGRD" | wgrib –i grib <gribfile> -o <newfile>
wgrib -s <gribfile> | grep "d=yymmddhh" | wgrib -i grib <gribfile> -o <newfile>

WGRIB-2:
To see everything in a Grib-2 archive:
wgrib2 <gribfile> or wgrib2 -v <gribfile> or wgrib2 -v2 <gribfile>
To see header and grid info do:
wgrib2 -V <gribfile>
To extract a certain field using grep and write to a new GRIB-2 file:
wgrib2 <gribfile> | grep ":UGRD" | wgrib2 <gribfile> -i -grib <newfile>
wgrib2 <gribfile> | grep "d=yyyymmddhh" | wgrib2 <gribfile> -i -grib <newfile>

---
wgrib v1.6.0: author: Wesley Ebisuzaki

"Wgrib" is a portable program to read grib files that were created
by the NCEP/NCAR Reanalysis Project. Of course, the program is not
restricted to Reanalysis files but Eugenia Kalnay is happy whenever
she sees the phrase "NCEP/NCAR Reanalysis".

The documentation for wgrib is spread over several files, readme, readme.dos, formats.txt, grib2ieee.txt, notice, porting.txt, tricks.wgrib and usertables.txt and changes.

Running wgrib without any arguments displays a short help message.

```
Portable Grib decoder for NCEP Operations etc.
  it slices, dices    v1.6.0 prelim 2 (7-01-97) Wesley Ebisuzaki
  usage: ./wgrib [grib file] [options]

Inventory/diagnostic output selection
  -s/-v/-V  short inventory/verbose inventory/very verbose non-inventory
  (default) regular inventory
Options for inventory/diagnostic output
  -PDS/-PDS10/-GDS/-GDS10    print PDS/GDS in hex/dec
  -verf                      print forecast verification time
  -4yr/-ncep_opn/-ncep_rean  see documentation

Decoding Grib selection
  -d [record number]         dump record number
  -p [byte position]         dump record at byte position
  -i                         dump controlled by stdin (inventory list)
  (none)                     no decode .. inventory only
Options for decoding Grib
  -text/-ieee/-bin/-grib     dump to a text/ieee/bin/grib file
  -h/-nh                     dump will have headers (default)/no headers
  -H                         dump will include PDS and GDS (-bin/-ieee only)
  -append                    append to dump file
  -o [file]                  output file name, 'dump' is default
```

*** Standard Inventory ***

WGRIB's first duty is create an inventory. This inventory also serves as an index file. Using the test file land.grb you should be able to enter:

```
% wgrib land.grb
Using NCEP reanalysis table, see -ncep_opn, -ncep_rean options
1:0:d=87010100:LAND:kpds5=81:kpds6=1:kpds7=0:TR=0:P1=0:P2=0:TimeU=1:sfc:anl:NAve
=1
```

The first line indicates that wgrib couldn't figure out whether to use the reanalysis or operational grib tables. Since land.grb is from reanalysis, we should use the reanalysis tables. Trying again, we get

```
% wgrib land.grb -ncep_rean
```

1:0:d=87010100:LAND:kpds5=81:kpds6=1:kpds7=0:TR=0:P1=0:P2=0:TimeU=1:sfc:anl:NAve
=1

The inventory consists of several fields separated by colons. The contents
of the fields are:

1. Record number
2. Position in bytes
3. Date (YYMMDDHH).
4. Parameter name (LAND=land/sea mask)
5. Indicator of parameter and units (grib PDS octet 9)
6. Type of level/layer (grib PDS octet 10)
7. Height, pressure, etc (grib PDS octets 11-12)
8. Time Range (grib PDS octet 21)
9. Period of time 1, (grib PDS octet 19)
10. Period of time 2, (grib PDS octet 20)
11. Forecast time unit (grib PDS octet 18)
12. level
13. anl=analysis, fcst=forecast
14. NAve (number of grids used to make average)

*** Short Inventory ***

The short inventory can be obtained using the -s option. This
inventory is easier to read the the previous inventory and can also
be used as an index file.

%wgrib -s land.grb -ncep_rean
1:0:d=87010100:LAND:sfc:anl:NAve=1

1. Record number
2. Position in bytes
3. Date (YYMMDDHH).
4. Parameter name (LAND=land/sea mask)
6. Type of level/layer (grib PDS octet 10)
7. Forecasts, analysis, etc
8. For an average, the number of fields averaged together

*** Verbose Inventory ***

The small verbose inventory can be obtained using the -v option.
This inventory can be used as an index file.

% wgrib -v land.grb -ncep_rean
1:0:D=1987010100:LAND:kpds=81,1,0:sfc:anl:"Land-sea mask [1=land; 0=sea]

1. Record number
2. Position in bytes
3. Date (YYYYMMDDHH).
4. Parameter name (LAND=land/sea mask)
5. KPDS5, KPDS6, KDPS7 (PDS Octets 9, 10, 11-12)
6. Type of level/layer (grib PDS octet 10)
7. Forecasts, analysis, etc
8. Description of parameter type

*** Verbose Description ***

The fourth type of file description can not be used as an index file.
However, it is more human readable. It gives you information that is not
normally available such as grid dimensions. Using the test file land.grb,
you should be able to enter:

%wgrib land.grb -V -ncep_rean

```
rec 1:0:date 1987010100 LAND kpds5=81 kpds6=1 kpds7=0 levels=(0,0) grid=255 sfc anl:
  LAND=Land-sea mask [1=land; 0=sea]
  timerange 0 P1 0 P2 0 TimeU 1  nx 192 ny 94 GDS grid 4 num_in_ave 1 missing 0
  center 7 subcenter 0 process 80 Table 2
  gaussian: lat  88.542000 to -88.542000
          long 0.000000 to -1.875000 by 1.875000, (192 x 94) scan 0 bdsgrid 1
  min/max data 0 1  num bits 4  BDS_Ref 0  DecScale 1 BinScale 0
```

The first line states
the record 1 starts at byte position 0
the initial date is January 1, 1987 at 00Z
the parameter is "LAND" (numeric code 81, PDS octet 9)
with a level type 1 (kdps6=1, PDS octet 10)
and value 0 (PDS octets 11-12)
or levels(0,0) (PDS octet 11, PDS octet 12)
with a user defined grid (grid=255)
and it is a surface analysis

The second line is a further description of the parameter type

The third line describes
timerange (PDS octet 21)
P1 (PDS octet 19)
P2 (PDS octet 20)
TimeU (PDS octet 14)
nx ny grid size as used by wgrib
GDS grid (GDS octet 6)
num_in_ave (PDS octet 22-23)

number missing from average (PDS octet 24)

The fourth line describes
center (PDS octet 5)
subcenter (PDS octet 26)
process (PDS octet 6)
parameter table version (PDS octet 4)

The fifth and sixth lines describe the grid type

The last line describes
minimum and maximum values of the data
the number of bits used to store the data
the minimum value
the decimal and binary scaling used

Most of the information within this description will only make sense if you have a copy of the GRIB definition as reference.

If you want to determine the contents of record N, try the command:

```
%wgrib land.grib -V -d N
```

This command also writes a binary dump of the record but it's quick. If you don't want a binary dump, try (on a UNIX machine),

```
%wgrib land.grib -V -d N -o /dev/null
```

*** Extracting Data ***

The second major function of wgrib is to extract data from a grib file. The output can be binary, IEEE (big endian), grib and text. All output formats except grib can be written with or without a header. See FORMATS.TXT for more information. The '-append' option appends the extracted data and the '-o [filename]' allows you to set the default output file which is normally "dump".

Note: binary format with a header is often compatible with fortran code.
Note: IEEE output is "big-endian".
Note: writing in binary is faster than writing ieee.
Note: using a binary format is faster, more precise and uses less disk space than the text format.

Note: The standard NCEP convention is that the arrays are stored in fortran order starting from the north and 0E. The following data goes south and eastward.

*** How to select data to be extracted ***

1) by record number
    wgrib land.grib -d 1   (extract first record)

2) by position
    wgrib land.grib -p 0   (extract record starting at byte 0)

3) by (machine readable) inventory (UNIX/AMIGA/MS-DOS)
    wgrib land.grb | wgrib -i land.grb -o output.bin

The third method is the most powerful one. Suppose you have a grib file with many different fields. You want to extract all the zonal winds (UGRD in NCEP files), you could type at a Unix machine:

    wgrib grib_file | grep ":UGRD:" | wgrib grib_file -i

Suppose you want to extract the 500 mb U winds, then you could type at a Unix machine:

    wgrib grib_file -s | grep ":UGRD:" | grep ":500 mb:" | wgrib -i grib_file

For more information on how to write ieee, binary, text and grib files see the file FORMATS.TXT.

*** ***

-

Suppose you wish to convert all the 500 mb heights (HGT in NCEP files) to binary with a header. The following line would convert "infile" to "outfile".

  % wgrib -s infile | grep ":HGT:500 mb:" | wgrib -i infile -o outfile

The "outfile" is often compatible with the fortran compiler.

Suppose you wish to convert all the 500 mb heights (HGT) to binary with a NO header. The following line would convert "infile" to "outfile".

  % wgrib -s infile | grep ":HGT:500 mb:" | wgrib -i -nh infile -o outfile

The "outfile" is often compatible with fortran direct-access I/O.

Converting a grib file into a text file is slow (reading and writing),
takes up much more disk space and can have less precision. Nevertheless
it has its uses.

```
% wgrib -s infile | grep ":HGT:500 mb:" | wgrib -i -text infile -o outfile
```

Most workstations computers use big-endian IEEE as their binary format.
For these machines, one should not use the -ieee option as it is slower
and could lose some precision. However, the following line will create
a big-endian IEEE with f77-style headers.

```
% wgrib -s infile | grep ":HGT:500 mb:" | wgrib -i -ieee infile -o outfile
```

Without headers, one would use

```
% wgrib -s infile | grep ":HGT:500 mb:" | wgrib -i -nh -ieee infile -o outfile
```

Suppose you have a large file with every variable imaginable. But you
are a simple person with limited means. You only want the 500 mb heights
and you have limited disk space. The following will extract the
500 mb heights as a grib file.

```
% wgrib -s infile | grep ":HGT:500 mb:" | wgrib -i -grib infile -o outfile
```

---

wgrib -> wgrib2

Converting scripts that use wgrib to wgrib2 should be straight forward.

-d all	(no options)
-d N	-d N or -d N.M (for grib2 submessages)
-bin -o FILE.BIN	-bin FILE.BIN
-text -o FILE.TXT	-text FILE.TXT
-ieee -o FILE.BIN	-ieee FILE.BIN
-grib -o FILE.GRIB	-grib FILE.GRIB
-nh	-no_header
-h	-header
-verf (sets verf time flag)	-verf (write inventory with verf time)
-s -verf	-verf
-V (verbose inventory)	-v (verbose) and use the following to get

	-grid (grid description) -bitmap (bitmap information) -stats (minumum/maximum value) -packing (to see packing information) -scan (to scan order)
-PDS/-PDS10	n/a
-GDS/-GDS10	n/a
-ncep_opn/-ncep_rean	n/a
-4yr	n/a
-ncep_ens	n/a
-p	n/a
-dwdgrib	n/a
-H	n/a
-o	not needed
-----	-order ???? (grids are converted to we:sn order by default) use -order we:ns for GFS, nothing for NAM.

Changed inventory format, different searches

The wgrib2 inventory has changed. The various grep/egreps will have to be changed to see if they are compatible with new inventory format

```
wgrib  FILE | grep ":HGT:" | wgrib -i  FILE -bin -o FILE.BIN
wgrib2 FILE | grep ":HGT:" | wgrib2 -i FILE -bin FILE.BIN
wgrib2 FILE -bin FILE.BIN -match ":HGT:"

wgrib -4yr FILE | grep ":d=2006081712:" | wgrib -i  FILE -bin -o FILE.BIN
wgrib2 FILE | grep ":d=2006081712:" | wgrib2 -i  FILE -bin -o FILE.BIN
wgrib2 -match ":d=2006081712:" FILE -bin -o FILE.BIN
```

wgrib2 uses a 4 digit year code. Scripts using 2 digit years need to modified.

convert:

```
wgrib  FILE | grep ":d=06081712:" | wgrib -i  FILE -bin -o FILE.BIN
wgrib2 FILE | grep ":d=2006081712:" | wgrib2 -i  FILE -bin -o FILE.BIN
wgrib2 -match ":d=2006081712:" FILE -bin FILE.BIN
```

wgrib2 doesn't print out kdps5 .. kpds7 which are not applicable to grib2.

convert:

```
wgrib  FILE | grep "kpds5=7:kpds6=100:kpds7=500:" | wgrib -i  FILE -bin -o FILE.BIN
wgrib2 -match ":HGT:" -match ":500 mb:" -bin FILE.BIN FILE
```

grep/egrep

When you use wgrib, you end up using lots of greps. Many of the greps will have

to be rewritten because the text has been altered (ex. "10 m above gnd" -> "10 m above ground"), are gone ("kpds5=6"), or replaced by a new format. GRIB2 is big compared with GRIB1 so things had to change. Generally the wgrib2 version use fewer abbreviations because it is easier to understand and line length is less of an issue with a flexiable inventory format. Scan, Order of the Data In wgrib, files were decoded in the "raw" order; i.e., the order that they were written. For most files the order was we:ns or we:sn. With GRIB2, the complexity of the order was increased. In order to make life easier for the user, wgrib2, by default, put the data in a we:sn order. (There is an option to put the data in a we:ns order.) Command line: order of options With wgrib, option processing was simple. You didn't care where the option went and if you had conflicting options, the last one was used. With wgrib, the flags chnaged the configuration before the processing of the grib data.

Wgrib2 is a much more dynamic program. Each option now runs a subroutine. As with subroutines, the order is now important and the subroutine can be called multiple times. These subroutines are run in followin order.

1) In command-line order with the mode set to Initialize.
(for each field)
2) In command-line order with mode set to Process and a copy of the data
(end of loop)
3) In command-line order with the mode set to Finalize

##
##

##
##
1. To prevent some array size limits from putting too much memory on the stack:
ulimit –s unlimited

2. To check CPU info:
cat /proc/cpuinfo

3. View mounted or pre-mounted hard drives for data storage:
fdisk –l

4. To mount drives:
mount </dev location> <path location>
Example: mount /dev/scd1 /media/disk

5. Cluster all-node command for RAMS:
sshall ` ps auxr | grep “rams” `

##

##

(-)

##
##

January
01-001 02-002 03-003 04-004 05-005 06-006 07-007 08-008
09-009 10-010 11-011 12-012 13-013 14-014 15-015 16-016
17-017 18-018 19-019 20-020 21-021 22-022 23-023 24-024
25-025 26-026 27-027 28-028 29-029 30-030 31-031
February
01-032 02-033 03-034 04-035 05-036 06-037 07-038 08-039
09-040 10-041 11-042 12-043 13-044 14-045 15-046 16-047
17-048 18-049 19-050 20-051 21-052 22-053 23-054 24-055
25-056 26-057 27-058 28-059
March
01-060 02-061 03-062 04-063 05-064 06-065 07-066 08-067
09-068 10-069 11-070 12-071 13-072 14-073 15-074 16-075
17-076 18-077 19-078 20-079 21-080 22-081 23-082 24-083
25-084 26-085 27-086 28-087 29-088 30-089 31-090
April
01-091 02-092 03-093 04-094 05-095 06-096 07-097 08-098
09-099 10-100 11-101 12-102 13-103 14-104 15-105 16-106
17-107 18-108 19-109 20-110 21-111 22-112 23-113 24-114
25-115 26-116 27-117 28-118 29-119 30-120
May
01-121 02-122 03-123 04-124 05-125 06-126 07-127 08-128
09-129 10-130 11-131 12-132 13-133 14-134 15-135 16-136
17-137 18-138 19-139 20-140 21-141 22-142 23-143 24-144
25-145 26-146 27-147 28=148 29-149 30-150 31-151
June
01-152 02-153 03-154 04-155 05-156 06-157 07-158 08-159
09-160 10-161 11-162 12-163 13-164 14-165 15-166 16-167
17-168 18-169 19-170 20-171 21-172 22-173 23-174 24-175
25-176 26-177 27-178 28-179 29-180 30-181
July
01-182 02-183 03-184 04-185 05-186 06-187 07-188 08-189
09-190 10-191 11-192 12-193 13-194 14-195 15-196 16-197
17-198 18-199 19-200 20-201 21-202 22-203 23-204 24-205
25-206 26-207 27-208 28-209 29-210 30-211 31-212
August
01-213 02-214 03-215 04-216 05-217 06-218 07-219 08-220
09-221 10-222 11-223 12-224 13-225 14-226 15-227 16-228
17-229 18-230 19-231 20-232 21-233 22-234 23-235 24-236
25-237 26-238 27-239 28-240 29-241 30-242 31-243
September
01-244 02-245 03-246 04-247 05-248 06-249 07-250 08-251

09-252 10-253 11-254 12-255 13-256 14-257 15-258 16-259
17-260 18-261 19-262 20-263 21-264 22-265 23-266 24-267
25-268 26-269 27-270 28-271 29-272 30-273
October
01-274 02-275 03-276 04-277 05-278 06-279 07-280 08-281
09-282 10-283 11-284 12-285 13-286 14-287 15-288 16-289
17-290 18-291 19-292 20-293 21-294 22-295 23-296 24-297
25-298 26-299 27-300 28-301 29-302 30-303 31-304
November
01-305 02-306 03-307 04-308 05-309 06-310 07-311 08-312
09-313 10-314 11-315 12-316 13-317 14-318 15-319 16-320
17-321 18-322 19-323 20-324 21-325 22-326 23-327 24-328
25-329 26-330 27-331 28-332 29-333 30-334
December
01-335 02-336 03-337 04-338 05-339 06-340 07-341 08-342
09-343 10-344 11-345 12-346 13-347 14-348 15-349 16-350
17-351 18-352 19-353 20-354 21-355 22-356 23-357 24-358
25-359 26-360 27-361 28-362 29-363 30-364 31-365