Updates in LTRANS v.2b

Ian Mitchell

June 11, 2013

The following changes were made to LTRANS v.2 to improve the code and in response to reported bugs:

- The paths in the code were updated to be non-specific to our system. For example, the include LTRANS.h in the parameter module now refers to the folder in which the code resides.
- Added readDens and constDens to LTRANS.h and LTRANS.data so that parameter module can access them.
- Changes were made involving comparisons for better standard compliance, mostly using .EQV. instead of ==.
- Changes were made involving write formatting for better standard compliance. Commas after write statements like this were removed: "WRITE(*,*), variable"
- Fixed the precision and type of some variables. There were a few cases where floating point numbers were used as loop counters and these were converted to integers. There were many implicit casts that were made explicit, or corrected the precision of so there was no cast. Some of the math needed to be changed to reflect this.
- Changed some of the explicit casts already in the code to explicitly use accurate rounding.
- Assigned all pointers to NULL on instantiation. Pointers in fortran have three possible statuses: undefined, associated, and non-associated. Attempting to test an undefined pointer for association will result in an error. Gfortran's default behavior was to instantiate pointers as undefined. Setting them to NULL allows for association testing.
- Fixed free slip code, and added a switch in LTRANS.data to turn it on or off.
- Chris Sherwood added explicit precision to the format of some write statements (apparently gfortran needed it).
- Removed some unused variables
- Added code to ensure that to keep the corrected value of particles pushed down by the sea surface for interpolation
- Updated the makefile to allow the use of both ifort and gfortran with flags that have the consistent results between compilers on our machine. (NOTE: NetCDF libraries need to be compiled with the compiler used for LTRANS. You cannot use gfortran NetCDF libs with ifort or vice versa.)

  For ifort we use: **-vec -report0 –fp -model precise -mcmodel=medium**

-vec -report0 tells the compiler to not print to stdout the vectorizer diagnostic messages
-fp-model precise tells the compiler to only implement the most conservative floating point optimizations
-mcmodel=medium allows the compiled program to place its code into the first 2 gb of memory, and does not restrict the memory usage of the data.

By default, ifort also uses -O2, and includes many more optimizations than gfortran. The best fit for this in gfortran was the following:

-march=native -ffast-math -fno-cx-limited-range -O3 -funroll-loops --param max-unroll-times=4 -ffree-line-length none

-march tells the compiler what kind of cpu to generate code for, allowing great, processor specific optimizations. A gfortran user should set this value explicitly to reflect their machines architecture.
-ffast-math enables optimizations that possibly do not conform with IEEE or ISO standards
-fno-cx-limited-range this prevents some of the more extreme optimizations of -ffast-math
-O3 engages many compiler optimizations
-funroll-loops enables an optimization known as loop unrolling.
--param max-unroll-times==4 allows a maximum of 4 unrollings. This prevents executable from becoming too large.
-ffree-line-length none tells tge compiler to use entire line of code regardless of length.

A list of most compiler flags for gfortran can be found http://gcc.gnu.org/onlinedocs/gcc-4.1.1/gcc/Optimize-Options.html and http://gcc.gnu.org/onlinedocs/gfortran/Fortran-Dialect-Options.html. A list for ifort can be found at http://software.intel.com/sites/products/documentation/doclib/stdxe/2013/composerxe/compiler/fortran-lin/index.htm