Summary of modifications to netCDF Fortran (4.4.2 dev) Fortran 2003 interfaces for UCAR netCDF developers

Richard Weed
Jan. 23, 2016

1.0 Introduction

I have made a series modification to the netCDF Fortran 2003 interfaces I developed several years ago to remove any explicit dependency on the NC_MAX_DIM constant for arrays passed to the C interfaces along with cleaning up some of the code and adding support for the nc_open_mem function. Removing the dependency on NC_MAX_DIM eliminates the possibility of segmentation or memory faults that could occur with the automatic arrays I had statically sized with NC_MAX_DIM. Some compilers will default to putting automatic arrays on the stack by default. Others will look at the size of the array and put it on the stack if its smaller than some threshold. Finally, a few will always put automatic arrays on the heap unless you tell it otherwise with a compiler switch. By replacing all of the automatic arrays with allocatable arrays, the memory for these arrays will always be placed on the heap. This mod also has the advantage of reducing memory requirements for the case were installers make a local modification to increase the size of NC_MAX_DIM to some very large number even if the actual number of dimensions used is much smaller that NC_MAX_DIMS. Most of the modifications were straightforward but necessitated the addition of a few new Fortran specific C routines to return a value that was used to set the allocation size of some arrays.

A second set of modifications include some general code cleanup to make some cosmetic changes to match my current coding style. For example, I previously had the KIND= keyword on all type definitions. I removed the majority of them since I find (now after several years of writing Fortran >=90 code) to be overkill. I also changed the length parameter I returned with my addCNullChar function to include the appended C_NULL_CHAR to the final length. This allowed me to remove a +1 I was adding to the substring length I was passing to the C routines. I also discovered an bug in the V2 interfaces that has been there since I first wrote the code (almost 10 years ago now). I was subtracting 1 from the stride values in the routines that use them. This was never detected because there are no test programs for the V2 interfaces that call these functions (ncvptg, ncvpcc, ncvgtg, ncvgcc).

A third set of mods were made to my routines that define the various error codes etc. to make them compatible with the values in the f90 interfaces netcdf_constants.f90 routine plus some updates to reflect the current C code values.

A final modification was made to provide support for the nc_open_mem function to create an “in memory” file option. Since I was uncertain based on what I read about the function in the netCDF C 4.4 documentation how the function was designed to be used, I assumed that one use would be to read an existing file into memory as a C (or Fortran) stream using standard C or Fortran I/O routines and pass a pointer to the memory containing the file to nc_open_mem. I implemented this in Fortran assuming that the netcdf file was first read into an array of C_CHAR (1 byte) values. This array would probably need to be allocated. Therefore, an explicit interface was created that should be used to pass the “in-memory file (ie. the array of C_CHAR values) to the underlying C routines to avoid the probability of the compiler trying to “copy-in” a very large block of memory instead of just passing a pointer. Without the interface, the compiler will assume the function (nf_open_mem) is an external routine and will probably decide it has to do a “copy-in”. Unfortunately, this makes creating a standalone nf90_open_mem a little problematic since it relies on the existence of the Fortran 2003 ISO_C_BINDING module that contains the C_CHAR kind definition and precludes using the old cfortran.h interfaces. Until I can decide a better approach for adding an nf90_ interface, I have provided the interface to nf_open_mem (contained in the file module_netcdf_nf_interfaces.f90) with an additional generic name (nf90_open_mem) so that a reference to nf90_open_mem is a direct call to nf_open_men. To support testing of nf_open_mem, I created a test program that I placed in nf03_test called f03tst_open_mem.F that uses the logic from f03tst_vars6.F to create a netCDF file that is then read into memory as a stream and then opened as an in-memory file and checked using the final checks found in f03tst_vars6.F. The Makefile.am file in nf03_test was modified to add the nf_open_mem test.

2.0 Summary of modified routines

All of my routines (the ones that start with module_ or nf_) were modified during the code cleanup. The following routines were changed to remove the NC_MAX_DIMS dependency : nf_fortv2.f90, nf_genvar.f90, nf_nc4.f90, nf_var1io.F90, nf_varaio.F90, nf_varmio.F90 and nf_varsio.F90. In addition, three new C routines were created and placed in nf_lib.c. These routines are name nc_inq_compound_field_ndims, nc_inq_numgrps, and nc_inq_numtypes and are used to return the number of dims_sizes, groups, and types for a defined ncid.

The new nf_open_mem function was placed in nf_control.F90 since I was unsure if it was a NETCDF4 only function or was used for both NETCD3 and NETCDF4. The C – interop interface for nc_open_mem was placed in module_netcdf_nc_interfaces.f90 and an explicit Fortran interface for nf_open_mem was defined in module_netcdf_nf_interfaces.f90.

Routine module_netcdf_nc_data.f90 and module_netcdf_nf_data.F90 were modified to add new NC and NF parameters for errors, types, modes etc.

3.0 Testing

All tests were made using netcdf-C-4.4 built with HDF5 1.8.16. openMPI 1.10.1 was used for parallel builds and tests. The modifications were made to the development (pre 4.4.3 release) version of netcdf-fortran from around Jan. 15, 2015. make check serial tests were run with the following compilers and Linux OS versions using workstations with Intel XEON or I7 processors

gcc/gfortran 5.3 with Linux Mint 17.3 MATE edition (AKA Ubuntu 14.0.4 LTS)
gcc/gfortran 4.8.5 with Linux Mint 17.3
Intel 14.0.1 (icc/ifort) with Linux Mint 17.3

gcc/gfortran 4.8.3 with Centos (RHEL) 6.5
Intel 15.0.2 (icc/ifort) with Centos 6.5
pgi-2014 (14.10) with Centos 6.5

All the standard serial make check tests ran successfully for the modified code for all of the above compilers and Linuxes. A single parallel test was run using gcc/gfortran 5.3 on Linux Mint 17.3. As with the serial tests, the parallel tests all ran successfully.

The new code to support “in-memory” files via a call to nc_open_mem was tested using the new f03tst_open_mem.F test program in the nf03_test directory. The resulting log file suggest it ran successfully. As stated previously, I took an existing test program (f03tst_vars6.F) and modified it to replace the reopening of the netcdf file created during the code via nf_open with a call to nf_open_mem that passes a pointer to a an array of C_CHAR values that I read the contents of the file created by the program (f03tst_open_mem.nc) into using an unformatted Fortran read statement with the file opened with STREAM access. This is followed in the code by a call to a check program that uses the ncid generated by nf_open_mem to access the data in the file.

4.0 Issues encountered with pre-4.4.3 development release build setup.

I encountered one issue with the configure/autoHell build process (didn't try Cmake). For some reason, I had to explicitly set F77 and FFLAGS along with FC and FCFLAGS to get the make checks to run. I think in the absence of an explicit F77 environment variable, make check should just default to FC. Frankly, I would remove all reference to F77 in the build process anyway. Unless someone is trying to build on a 20 year old SGI with an MIPS compiler, there is no such thing as a pure F77 compiler anymore at least one you would want to use. Some compilers do have a -f77standard like switch that effectively turns off all F90 and greater features but why would you want to do that since most (probably all) modern Fortran compilers will compile old F77 code as good as or better than the original F77 compilers will . I also modified the shell scripts that run the parallel tests to replace the explicit mpiexec command with a $MPIEXEC environment variable that I set to the path to mpiexec (ie)

```
export MPIEXEC=/opt/openMPI/bin/mpiexec
```

I've found over the years this is much safer than hardwiring an explicit path in a shell script and I suggest UCAR consider a similar strategy as part of the build and test process.

5.0 Suggested Mods

One mod I would suggest is to merge the definition of the default KIND parameters defined currently defined in typeSizes.f90 with the way I define equivalent values in module_netcdf_nc_data.F90 using the Fortran 2008 intrinsic parameters if supported by the compiler or define equivalent parameters using selected_int and selected_real kind functions that have the same names as the Fortran 2008 values. I suggest that my logic be moved into typeSizes.f90 (renamed typesizes.F90 to allow pre-processing and my logic in module_netcdf_nc_data.F90 be replaced with a USE typeSizes. This will require typeSizes.F90 be the first thing compiled (before all the other netcdf, module_ and nf_ routines). I would have done this but I was in no mood to dive into autoHell and hack Makefile.am etc. However, I'm including modified versions of typeSizes.f90 and my module_netcdf_nc_data.F90 that implement this mod (see SUGGESTED_MODS subdir)

6.0 Possible Future mods

One thing missing due to a lack of a good way to support it in F90 or F77 is support for varying string arrays etc. Fortran 2003/2008 and soon to be 2015 open up several ways to implement this but its going to take some thought as to what is the best way to do it in the long run. You could do something now in Fortran 90 with arrays of fixed length strings but that is probably not what you want. Therefore, to do it right you will need to use something like an array of C_PTRs that contain the C address of deferred length string (Fortran 2003 way of doing varying length strings). This is probably the closest thing to the char **val array pointer definition in the C interfaces. The ultimate future mod would be a Fortran 2008/2015 object oriented package that (if designed correctly) could drastically reduce the amount of code but that as with all OO projects requires you get the design correct from the start or its not worth the effort.

I'll continue to provide what support I can for my code but unfortunately my time to work on this is pretty limited so I can't guarantee I can help with anything other than fixing obvious errors. However, I will try to answer any questions about this set of mods and other Fortran issues.

RW
Jan. 2016