ACME MMF Standalone Framework

Information

The ACME MMF standalone code lives in ACME/components/cam/src/physics/crm/standalone, and it essentially hijacks the crm(...) subroutine in crm_module.F90.

The standalone code lives in place in the ACME-ECP codebase so that the standalone model never becomes stale relative to ACME-MMF development.

There is a standalone/src/crm_standalone.F90 code that reads in data dumped out during actual runs and then makes calls to crm(...) with that data. Overall, the standalone code only needs to compile on the order of 60 files, takes only one minute to fully compile from scratch on Titan, and it is completely decoupled from the ACME and CAM infrastructure.

There are several places the standalone framework is severed from CAM and ACME in the crm_module.F90 code, and this is activated by the -DCRM_STANDALONE CPP define during compile time. Mostly it takes data from the input file instead of obtaining it from the ACME infrastructure.

The standalone code is inherently tied to data files that are dumped out during runtime in the file crm_dump.F90. The reasons the standalone model is tied to an output file are (1) you have to have realistic data to drive the standalone model in the first place and (2) by configuring your standalone build using a data file, you guarantee that your standalone executable is compatible with the data you're using for it (i.e., crm_nx, crm_dt, PLEV, etc.).

Checking CRM Standalone Answers

In general, the answers you get from CRM standalone will **not** be the same as the answers you get from the main model, but I have verified that they are close. The reason is that realistic runs, you end up with far more CPPDEFS that activate code that affects the output but not the portions of the code we're currently porting – things like CLUBB hooks, MODAL_AERO, and others. These macros activate code that significantly increases the dependencies for the standalone model that would make the compile time longer and thus make the standalone model less useful. So standalone output is to be compared against other standalone output, not output from the actual model.

The best way to check answers is to use the ./compute_diffs.sh utility that is automatically copied to your standalone build directory. It's a wrapper around an NCL script. I recommend running with -O2 and -O0 and seeing what the diffs are. When porting to GPUs or other accelerators, or making code modes for performance in general, the diffs should be on par with the -O0,-O2 diffs. The NCL wrapper script checks for all output variables.

TODO

I don't have this implemented yet, but I plan to provide a distribution (min,Q2,median,Q3,max) for the errors across all of the columns for each of the variables. Right now, you get a single error norm over all the columns, which is probably sufficient, but it's not really enough in my opinion. We don't want a large L-inf error hiding in the averages.

Dumping Out Data For Standalone Driver

In order to dump this data yourself, you simply need to specify -DCRM_DUMP in the CPPDEFS in the Macros file before you compile, and it will dump out samples of input and output data for calls to crm(...) in standalone mode.

When creating your own driver data using CRM_DUMP, you can (and probably should) also specify -DCRM_DUMP_RATIO=[ratio]. For instance if you say -DCRM_DUMP_RATIO=0.01, it will randomly select only roughly one out of every hundred crm(...) instances for saving data. It significantly decreases the amount of output you save while also maintaining a pretty representative sampling.

The crm_in.nc contains the data used for input, intent(in), to the crm(...) routine. The crm_out.nc file contains data that was output, intent(out), from the crm(...) routine. Both of these files are created by: (1) running an ACME-MMF run with -DCRM_DUMP specified and then (2) combining the multiple processes' files with "ncrcat crm_in_*.nc crm_in.nc" and likewise for crm_out.nc Then these files can be read in by crm_standalone.F90 to drive the crm(...) routine.

Already-Generated Example Data

I've run an FC5AV1C-L compset at ne16np4 resolution with 72 (58) vertical levels for ACME (CRM), 1km horizontal grid spacing, and 32 horizontal grid points with all the FC5AV1C-L physics turned on and 2-moment microphysics. You can find the data in /ccs/home/imn/crm_in.nc. There are currently 3272 columns stochastically generated from varying columns at varying times throughout a 5-day simulation. The typical

runtime on Titan using one node (i.e., "aprun -np 16 ./crm_standalone crm_in.nc") is 90-120 seconds. Soon, I will reduce the size of this dataset for even faster runs, but I think this is reasonable, and I'm pretty sure it will exercise all of the code we plan to port.

How To Use

A typical workflow for building the CRM is as follows:

```
cd standalone
source environments/env_[MACH]_[COMPILER].sh
cd utils
./setup_from_file.sh -crm_root /path/to/ACME-ECP/components/cam/src/physics/crm -build_root
/path/to/build/root -file /path/to/dumped/input/to/crm/crm_in.nc
cd /path/to/build/root
make -j32
mpirun -np [#procs] ./crm_standalone [/path/]crm_in.nc [/path/]output_prefix
./collect_files.sh [/path/]output_prefix
./compute_diffs.sh standalone_output_1.nc standalone_output_2.nc
vim timing_stats
```

The data dumping utility I use dumps out one file per process, so they have to be collected into a single file using "collect_files.sh," which is basically a wrapper around ncrcat from the NCO utilities.

On Titan, you cannot run executables interactively unless you are on an interactive qsub job. So I recommend placing your build_root in lustre project space and then compiling and running inside an interactive job. Remember you'll have to run "aprun -n [#procs] ./crm_standalone crm_input_file output_prefix" on Titan.

Software Dependencies

The CRM standalone model and its utilities require NetCDF (strictly required), NCO (to concatenate files), and NCL (to compute diffs).

Directories

Environments

In the crm/standalone/environments directory reside files you need to source to get NetCDF in your path, to get the NCO and NCL modules loaded (or placed in the PATH), and to set any custom CPPDEFS for your run. For the Makefile, you need to at least specify:

```
export FC=mpif90
export CC=mpicc
export FFLAGS="-O2"
export CFLAGS="-O2"
export FREEFLAGS="-ffree-line-length-none"
export FIXEDFLAGS=""
export LDFLAGS="${NCLIBS}"
export INCLUDE="${NCINC}"
export CPPDEFS=" -DFORTRANUNDERSCORE "
```

Utils

The only utility you should have to directly interact with here is setup_from_file.sh. Everything else is either used or copied to your build directory for you to use there.

Src

There are only three source files in here. One to hijack abort_mp in cam_abort_utils so we don't have to use the actual file. Another hijacks time_manager.F90 because it doesn't actually matter for crm_standalone. Finally, there's the crm_standalone.F90, which is the main driver for the CRM standalone, which reads in the input data and outputs data from the standalone runs.