Running gcov coverage analysis and profiling tool

To discover which portions of code were never executed, or to more generally profile a code

Step-by-step guide

NOTE: These instructions assume gcc/gfortran compilation environment (gcov does not work with other compilers), and NOT using containers. Though if using a container, likely some of the steps outlined below could simply be skipped. Also, in my case I was specifically interested in coverage analysis for the CRTM. For other codes such as FV3, other bundles will be needed.

NOTE also: The CRTM code is pure Fortran, so some of the instructions below such as editing compiler flags will need to be modified and/or augmented if C or C++ analysis is desired.

NOTE also: Somehow after bulleted items below, numbering starts back at one. I don't know what's going on there, but if someone more familiar with wiki editing knows how to fix it, please feel free to do so.

1. Clone the ufo-bundle; cd into the top-level directory.
2. **% git checkout feature/buildtools** Soon this branch should be merged into the "develop" branch. But for now it contains extra scripts needed for building with gcc/gfortran outside of a container. If building inside a container, this step can probably be skipped.
3. cd into "tools/", then edit the file **module_setup_linuxpc_gcc_openmpi.sh,** and modify as appropriate for your Linux system. If using a container, this step can be skipped.
   Still in the "tools/" directory, run the command: **% build.sh linuxpc gcc openmpi debug** The "debug" portion is critical, as gcov needs the flags associated with a "debug" build.
4. Hit "cntl-c" after the text **"-- Generating done"** appears. This will ensure that all required bundles are downloaded, but we need to kill the build because we need to modify some of the compilation flags. Don't worry if some files actually get compiled before hitting "cntl-c": The files will get recompiled later.
5. Edit top-level **CMakeLists.txt** file: Before commands that start "ecbuid_bundle" add the line: **link_libraries(gcov)** This will ensure that appropriate libraries required by the gcov tool are added at link time.
6. Edit files containing appropriate compilation flags. For my CRTM analysis, this meant files **ufo/cmake/compiler_flags_GNU_Fortran.cmake** and **crtm/cmake/compiler_flags_GNU_Fortran.cmake** Add to entry CMAKE_Fortran_FLAGS_DEBUG the flags: **-fprofile-arcs -ftest-coverage** T hese flags are required by gcov in order for it to work properly. For C/C++ analysis, you'll need to find in which files the appropriate flags are set, but gcc and gfortran both accept these same two flags.
7. Now cd back into the "tools/" directory and rerun **build.sh linuxpc gcc openmpi debug** to complete the build with appropriate flags set. The "debug" portion is critical, as gcov needs the flags associated with a "debug" build. Also: if desired an additional "threads" flag can be included after **debug** to specify number of threads to use for a parallel make. The default is 1. Hopefully it will run to completion and produce the executable(s) needed for coverage analysis/profiling.
8. Do the appropriate run. For my CRTM test this meant:

   - **% cd build_linuxpc_gcc_openmpi_debug/ufo/test**
   - **% test_ufo_amsua "--" "testinput/amsua.jso**

1. cd to where the .o files live that you want to profile. For me this was:

   - **% cd build_linuxpc_gcc_openmpi_debug/crtm/libsrc/CMakeFiles/crtm.dir**

1. Unfortunately, at least for CRTM, the compilation command does not use standard naming for .o files, e.g. instead of CRTM_x.f90 compiling to CRTM_x.o, it actually compiles to CRTM_x.f90.o This confuses gcov, so we need to rename the gcov-specific files which were created by the run so that when gcov is run, it will find what it is looking for. gcov-generated files are named *.gcda and *.gcno What I did to get around this problem was to write a script to provide soft link names that gcov will understand:

- **for i in $(ls *.f90.gcda *.f90.gcno); do**
- **newname=$(echo $i | sed -e 's/\.f90\././1')**
- **if [! -f $newname]; then**
- **ln -s $i $newname;**
- **fi**
- **done**

1. Next, gcov needs the source files to live in this same directory, which unfortunately they do not. So, again for CRTM, I needed to do this (still in the directory where the .o files live):

- **% ln -s ../../../../../crtm/libsrc/*f90 .**

1. Coverage analysis is per-file, and contained in files named *.gcov. So for example I was most interested in CRTM_K_Matrix_Module.f90, so looked at CRTM_K_Matrix_Module.f90.gcov Inside this file on the left hand side of each line are execution counts for each line of source code. The most important marks are:

- "#####" Five hash marks means the line did compile to executable code, **but was never actually executed.** This is what is most critical to coverage analysis. Either input data and/or control structures will need to be modified in order to ensure that these lines do get executed, if it is important that they do get executed. In CRTM there are a number of tests that exit upon failure. Since these lines are never executed, that is probably fine and no additions are necessary since they represent error conditions.
- "-" A dash means that the line did not compile into executable code. Comment lines are an example.
- "<some_number>" Numbers are a count of the number of times the line did get executed. From my cursory analysis, these numbers appear to be an **estimate** of the number of times it got executed, not a precise value.

Related articles

- Running gcov coverage analysis and profiling tool