Concurrent Single-Executable CCSM with MPH Library

Yun (Helen) He and Chris Ding
Computational Research Division
Lawrence Berkeley National Laboratory
May 2006

Abstract

Community Climate System Model (CCSM) is currently a multi-executable system based on the Multi-Program Multi-Data (MPMD) mechanism. Each component is compiled into a separate executable. MPMD is normally cumbersome in usage and vendor support is sometimes limited or completely unavailable, such as on BlueGene/L. Also smaller groups and institutions would like to run CCSM locally, rather than relying on large computer centers. So, single-executable CCSM is under request.

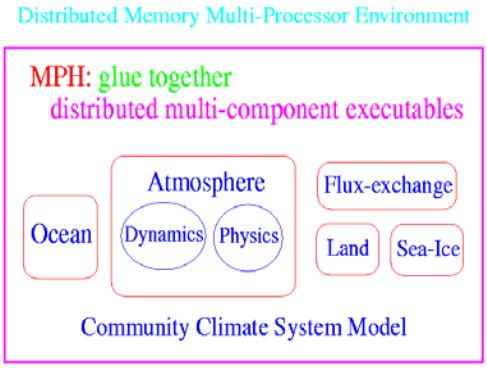
We are developing a multi-executable and single-executable coexisting version of CCSM. In single-executable, each component is organized as a subroutine, which is called from a master program. Different components run simultaneously. This is accomplished by redesigning the top level CCSM structures using the Multi-Program Handshaking (MPH) library.

Background Information

This single-executable Community Climate System Model (CCSM) [1] work is part of the Scientific Discovery through Advanced Computing (SciDAC) climate project titled "SciDAC Collaborative Design and Development of the Community Climate System Model"[2]. This project is a collaborative effort involving National Center for Atmospheric Research (NCAR), multiple DOE labs (ANL, LANL, LLNL, LBL, ORNL and PNL), and NASA/Data Assimilation Office (DAO). The goal of the project is to provide US researchers with state-of-the-art coupled climate simulation capabilities. CCSM is the U.S. flagship coupled climate model system most widely used in long-term climate system modeling in the U.S. It is publicly available to the climate community. CCSM modeling results are part of the US's submission to the intergovernmental Panel on Climate Change Fourth Assessment Report (IPCC AR4).

MPH Introduction

The single-executable CCSM is implemented using the MPH library. We briefly describe MPH here. Many large-scale simulation codes consist of several components that are developed by different groups and perform (semi) independent tasks. A multi-executable code integrates these components together as a single computational system, while keeping each component as a stand-alone executable. An example is the original CCSM, the coupled climate system consists of separate programs for modeling atmosphere, ocean, sea ice and land surface and a flux coupler linking the four components.



MPH [3] provides a key infrastructure for integrating separate executables together. It provides functionalities for component name registration and resources allocation, and it initializes communication channels between independent components. MPH provides a flexible, versatile mechanism for these tasks and is a basis for larger software tools/frameworks with multi-component approach.

MPH is adopted in many models for coupling model components, including Model Coupling Toolkit (MCT) [4], Coupler Component (CPL6) [5] in CCSM, National Center for Atmospheric Research (NCAR) MM5 [6] / WRF [7] models, Colorado State University (CSU) Geodesic Grid Model [8]). MPH also facilitates multi-instance ensemble simulations (with each instance as a model component within an executable), used by applications in European Centre for Medium-Range Weather Forecasts (ECMWF), the Edinburgh Parallel Computing Centre (EPCC) and many others include a Monte Carlo code running on 1024 processors. In this study, MPH is used for developing single-executable CCSM.

MPH Main Functionalities

- Flexible component name registration
- Run-time resource allocation
- Inter-component communication
- Query multi-component environment
- Stand-out / stand-in redirect
- Supports five model integration execution modes:
    - Single-Comp exec Single-Exec system (SCSE)
    - Multi-Comp exec Single-Exec system (MCSE)
    - Single-Comp exec Multi-Exec system (SCME)
    - Multi-Comp exec Multi-Exec system (MCME)
    - Multi-Instance exec Multi-Exec system (MIME)
- Easily switch between different modes
- Very small overhead in time and memory

Single-Executable CCSM Design Goals

• Produce a single-executable, concurrent CCSM
• Release version CCSM3.0
• Developmental version CCSM3.1
    • POP2 + datm7 + dice6 + dlnd6 + cpl6
    • POP2 + fv-CAM3.1 + developmental ice, lnd and cpl versions
• Coexist with multi-executable, concurrent CCSM
• Any combination of model components (dead, data, active)
• Use CCSM scripts
• On all CCSM supported platforms and other machines, such as Blue Gene/L

Resolved Technical Issues

• **Master program**

For single-executable, a master program is added for establishing the distributed multi-component environment via an MPH library call.

Master.F:

```
master_World=MPH_components_setup(name1="atm", name2="ice",  name3="lnd",
&                                 name4="ocn", name5="cpl")
if (Proc_in_component(“atm", comm))  call ccsm_atm()
if (Proc_in_component(“ice", comm))   call ccsm_ice()
if (Proc_in_component(“lnd", comm))  call ccsm_lnd()
if (Proc_in_component("ocn", comm))  call ccsm_ocn()
if (Proc_in_component(“cpl", comm))  call ccsm_cpl()
```

• **Redesign top level CCSM structure**

Only a minor redesign of top level source code is necessary to switch between MPMD and SPMD modes.

• **Coexistence of multi-executable and single-executable CCSM**

• **Flexible switching among different model options: active model, data model, and dead (mock) model.**

See the following pseudo codes for above three issues.

Subroutinized program structure:

```
#ifdef   SINGLE_EXEC
         subroutine ccsm_atm()
#else
```

```
        program ccsm_atm
#endif

        if (model_option = dead) call dead("atm")
        if (model_option = data) call data()
        if (model_option = active) call cam2()

#ifdef  SINGLE_EXEC
        end subroutine
#else
        end program
#endif
```

• **Revise CCSM build scripts**

• **Set OpenMP threads dynamically**

Allow OpenMP threads to be set to different values on different components. For single-exec, OpenMP threads are set from each component dynamically at run time (instead of environmental variables for multi-execs).

```
call MPH_get_argument("THREADS", nthreads)
call OMP_SET_NUM_THREADS(nthreads)

"processors_map.in":
atm 0 2 THREADS=4 file_1= xyz alpha=3.0 ...
ocn 3 5 THREADS=2
```

Name Conflicts Issue

Developed by different groups and institutions, different component models can easily have subroutines with the same names but different uses. Each subroutine name now becomes a global symbol name. The compiler may generate warnings for multiple matches but always uses the first match.

There are two probable solutions. One is the extensive renaming in the source codes, i.e., renaming all functions, subroutines, interfaces, and variables by adding a model prefix. It is tedious and also error-prone.

We propose an alternative "module-based approach" by using a wrapper module and "use module only" renaming on the fly. The key idea is localization of global symbols. The wrapper module is used to hide standalone subroutines of a component model so they become local entities of the wrapper module, and no global names are generated for these subroutines. The wrapper "includes" all the subroutines; in practice each subroutine is still a single file. Compile the main driver with all the modules, but no subroutines are explicitly referenced. There is minimal renaming only when different component modules are used in a same file. This solution is less tedious than adding a prefix to all subroutine names.

It is worth noting that common module names from different components do not cause name conflicts. Also, common subroutine names contained in different named modules do not cause name conflicts. However, when there are common subroutine names contained in the same module name from different components, the above module-based approach cannot solve the name conflicts, and the module names need to be manually renamed.

ocn has:	atm has:
ocn_main.F	*atm_main.F*
ocn_mod1.F	*atm_mod1.F*
ocn_sub1.F	*atm_sub1.F*
sub2.F	*sub2.F*

ocn_wrapper.F:

```
        module ocn_wrapper
        contains
# include "ocn_sub1.F"
# include "sub2.F" ! Local symbol
        end module
```

==========

add in "ocn_main.F": use ocn_wrapper

Renaming conflict names on the fly

This also works for variables and interfaces.

Suppose subroutine dead() is defined in both ocn_mod and atm_mod:

use ocn_mod, only: ocn_dead ➔ dead
use atm_mod, only: atm_dead ➔ dead
if (proc_in_ocn) call ocn_dead() ! instead of dead
if (proc_in_atm) call atm_dead() ! Instead of dead

Single-Executable Status

• Single-executable CCSM already works on NERSC IBM SP3 (Seaborg), and NCAR IBMs SP4 (Bluesky), NCAR IBM SP5 (Bluevista), and ORNL Cray-X1 (Phoenix).

• Both modified single-executable and modified multi-executable CCSM generates bit-for-bit with original multi-executable CCSM for all model configurations.

• Will port onto BlueGene/L, SGI Altix (possible), new NCAR production systems, and other CCSM supported platforms.

• Preliminary performance data shows run time within 10% difference from original multi-executable CCSM runs.

Related Web Pages

• http://hpcrd.lbl.gov/SCG/acpi/SE
• http://swiki.ucar.edu/ccsm/11
• http://hpcrd.lbl.gov/SCG/acpi/MPH
• http://www.ccsm.ucar.edu/models/ccsm3.0
• http://www-unix.mcs.anl.gov/mct/

Acknowledgements

This work is in collaboration with Mariana Vertenstein, Nancy Norton, Brian Kauffman, Anthony Craig, and Jon Wolfe from National Center for Atmospheric Research. It is supported by a DOE SciDAC climate project. This work uses the computational resources from NERSC, NCAR, and ORNL.

References

[1] Community Climate System Model. http://www.cgd.ucar.edu/csm

[2] SciDAC Collaborative Design and Development of the Community Climate System Model. http://www.scidac.org/CCSM.

[3] Y. He and C. Ding, Coupling Multi-Component Models with MPH on Distributed Memory Computer Architectures, International Journal of High Performance Computing Applications, Vol.19, No.3, 329-340, August 2005.

[4] J. Larson, R. Jacob, and E. Ong, The Model Coupling Toolkit: A New Fortan90 Toolkit for Building Multiphysics Parallel Coupled Models, International Journal of High Performance Computing Applications, Vol.19, No.3, 277-292, August 2005.

[5] A.P. Craig, R.L. Jacob, B. Kauffman, T. Bettge, J. Larson, E. Ong, C. Ding, and Y. He, CPL6: The New Extensible, High-Performance Parallel Coupler for the Community Climate System Model, International Journal of High Performance Computing Applications, Vol.19, No.3, 309-327, August 2005.

[6] MM5 Community Model Homepage. http://www/mmm.ucar.edu/mm5.

[7] The Weather Research and Forecasting Model Website. http://www.wrf-model.org.

[8] Overview of the BUG model. http://kiwi.atmos.colostate.edu/BUGS