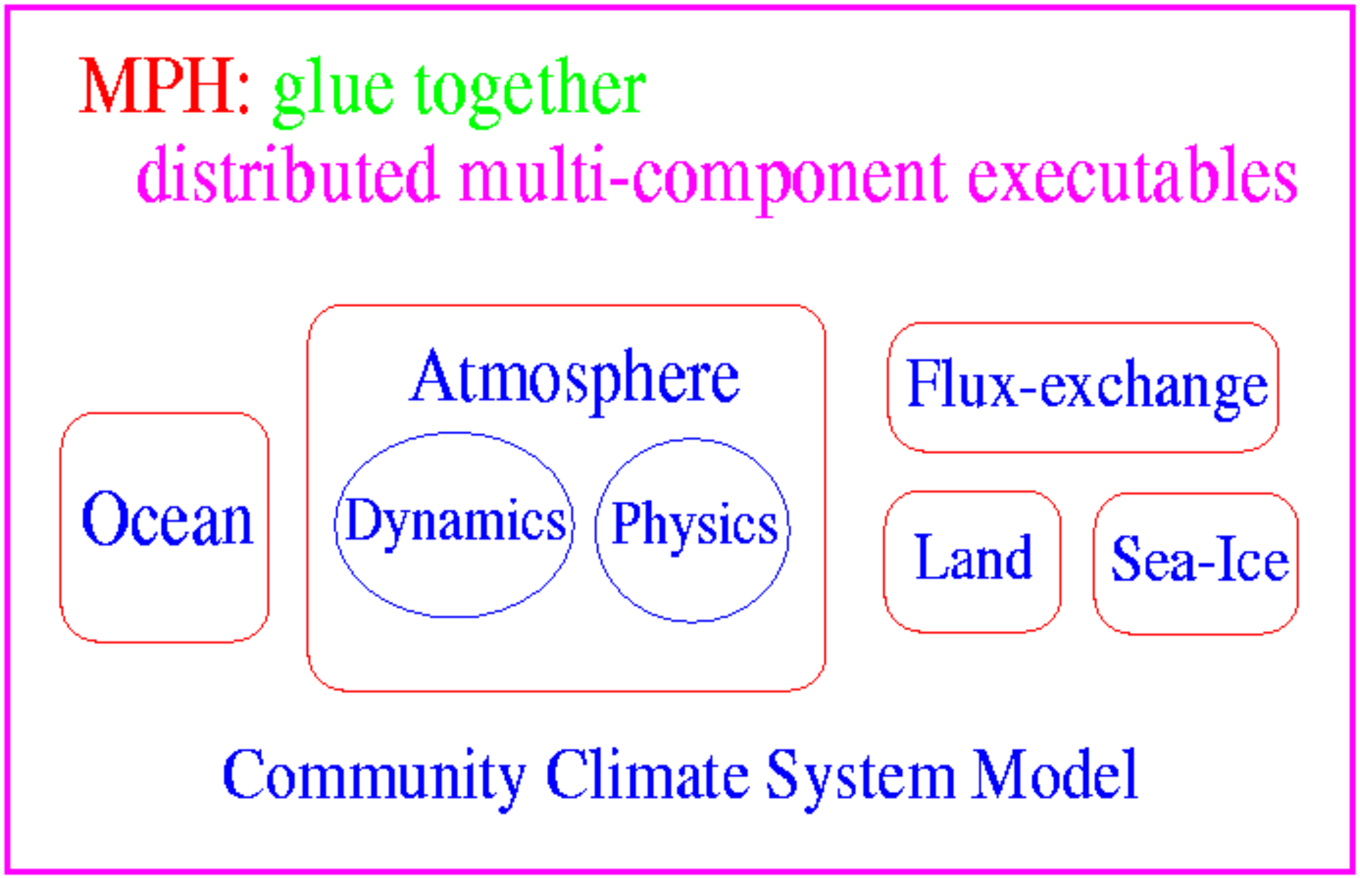
MPH: a Library for Coupling Multi-Component Models on Distributed Memory Architectures and its Applications

Yun (Helen) He and Chris Ding
CRD Division
Lawrence Berkeley National Laboratory

Distributed Memory Multi-Processor Environment



Motivation

- Application problems grow in scale & complexity
- Effective organization of simulation software system that is maintainable, reusable, sharable, and efficient → a major issue
- Community Climate System Model (CCSM) development
- Software lasts much longer than a computer!

Multi-Component Approach

- Build from (semi-)independent programs
- Coupled Climate System = Atmosphere + Ocean + Sea-Ice + Land-Surface + Flux-Coupler
- Components developed by different groups at different institutions
  - Maximum flexibility and independence
  - Algorithm, implementation depends on individual groups, practicality, time-to-completion, etc.
- Components communicate through well-defined interface data structure.

Distributed Components on HPC Systems

- Use MPI for high performance
- MPH: Multiple Program-Component Handshaking
  - MPI Communicator for each component
  - Component name registration
  - Resource allocation for each component
  - Support different job execution modes
  - Stand-out / stand-in redirect
  - Complete flexibility

A climate simulation system consists of many independently-developed components on distributed memory multi-processor computer

- Single-component executable:
  - Each component is a stand-alone executable
- Multi-component executable:
  - Several components compiled into an executable
- Different model integration modes:
  - Single-Component executable Single-Executable system (SCSE)
  - Multi-Component executable Single-Executable system (MCSE)
  - Single-Component executable Multi-Executable system (SCME)
  - Multi-Component executable Multi-Executable system (MCME)
  - Multi-Instance executable Multi-Executable system (MIME)

Component Integration / Job Execution Modes

- Multi-Component exec. Single-Executable system (MCSE):
  - Each component is a module
  - All components compiled into a single executable
  - Many issues: name conflict, static allocations, etc.
  - Data input/output
  - Stand-alone component
  - Easy to understand and coordinate

Component Integration / Job Execution Modes

- Single-Component exec. Multi-Executable system (SCME):
  - Each component is an independent executable image
  - Components run on separate subsets of SMP nodes
  - Max flexibility in language, data structures, etc.
  - Industry standard approach
  - Job launching not straightforward

Component Integration / Job Execution Modes

- Multi-Component exec. Multi-executable system (MCME):
  - Several components compiled into one executable
  - Multiple executables form a single system
  - Different executables run on different processors
  - Different components within same executable could run on separate/overlap subsets of processors
  - Maximum flexibility
  - Includes MCSE and SCME as special cases

Component Integration / Job Execution Modes

- Multi-Instance exec. Multi-executable system (MIME):
  - Same executable replicated multiple times on different processor subsets
  - Run multiple ensembles simultaneously as a single job
  - Ensemble statistics able to run on the fly
  - Dynamic control of future simulation
  - Efficient usage of computer resource

Multi_Instance Ensembles Example

- Multi-instance exec: 100 CCM ensembles
  - Embarrassingly parallel
- Multi-instance exec: 4 ocean ensembles
  one single-comp exec: statistics.
- Multi-instance exec: 3 atm ensembles
  one single-comp exec: ocn

Multi-Component Single-Executable (MCSE)

PCM

master.F:

```
 mph_exe_world = MPH_components_setup (name1= 'atmosphere' ,
&                                  name2= 'ocean' ,  name3= 'coupler' )

 if (Proc_in_component ( 'ocean' , comm)) call ocean_v1 (comm)
 if (Proc_in_component ( 'atmosphere' , comm)) call atmosphere (comm)
 if (Proc_in_component ( 'coupler' , comm)) call coupler_v2 (comm)
```

Component registration file:

```
BEGIN
Multi_Comp_Start
atmosphere      0    7
ocean           8   13
coupler        14   15
Multi_Comp_End
END
```

Single-Component Multi-Executable (SCME)

CCSM

Coupled System = Atmosphere + Ocean + Flux-Coupler

atm.F: atm_world = MPH_components_setup ("atmosphere")
ocean.F: ocn_world = MPH_components_setup ("ocean")
coupler.F: cpl_world = MPH_components_setup ("coupler")

Component Registration File:

```
BEGIN
atmosphere
ocean
coupler
END
```

Multi-Component Multi-Executable (MCME)

Most Flexible

exe1_world = MPH_components_setup (name1= 'ocean' , name2= 'ice')

exe2_world = MPH_components_setup (name1= 'atmosphere' ,
& name2= 'land' , name3= 'chemistry')

Component Registration File:

```
BEGIN
coupler                 ! a single-component executable
Multi_Comp_Start        ! first multi-component executable
ocean       0    15
ice         16   31
Multi_Comp_End
Multi_Comp_Start        ! second multi-component executable
atmosphere    0   15
land          0   15
chemistry     16  31
Multi_Comp_End
END
```

Multi-Instance Multi-Executable (MIME)

Ensemble Simulations

Ocean_world = MPH_multi_instance ("Ocean")

Component Registration File:

```
BEGIN
Multi_Instance_Start          ! a multi-instance executable
Ocean1     0    15    infile_1  outfile_1  logfile_1  alpha=3 debug=off
Ocean2    16    31    infile_2  outfile_2  beta=4.5  debug=on
Ocean3    32    47    infile_3  dynamics=finite_volume
Multi_Instance_End
statistics                    ! a single-component executable
END
```

Up to 5 strings in each line could be appended for passing parameters:

```
call MPH_get_argument ("alpha", alpha)
call MPH_get_argument(field_num=2, field_val=output_file)
```

Joining two components

- MPH_comm_join ("atmosphere", "ocean", comm_new)
  - comm_new contains all procs in "atmosphere", "ocean".
  - "atmosphere" procs rank 0~7
  - "ocean" procs rank 8~11
- MPH_comm_join ("ocean", "atmosphere", comm_new)
  - "ocean" procs rank 0~3
  - "atmosphere" procs rank 4~11
- Afterwards, data remapping with "comm_new"

Inter-Component communications

atmosphere sends message to ocean local_id= 3:

MPI_send (…, MPH_global_id (“ocean”, 3), MPH_Global_World,…)

MPH Inquiry Functions

- MPH_global_id()
- MPH_comp_name()
- MPH_total_components()
- MPH_exe_world()
- MPH_num_ensemble()
- MPH_get_strings()
- MPH_get_argument()
- …

Multi-Channel Output

- Normal standard out
  - print *, write(*,*), write(6,*)
- Need each component writes to own file
- Some parallel file system has “log” mode
- MPH resolves standard out redirect with the help of system function "getenv" or "pxfgetenv"
  - setenv ocn_out_env ocn.log
  - call MPH_redirect_output (comp_name)

Sample Job Script

```
#! /usr/bin/csh -f
# @ output = poe.stdout.$(jobid).$(stepid)
# @ error = poe.stderr.$(jobid).$(stepid)
# @ wall_clock_limit = 1800
# @ class = debug
# @ job_type = parallel
# @ node = 1
# @ total_tasks=14
# @ network.MPI = csss, shared, us
# @ queue

setenv MP_PGMMODEL mpmd
setenv MP_CMDFILE tasklist
setenv MP_STDOUTMODE ordered
setenv MP_INFOLEVEL 2

setenv ice_out_env ice.log
setenv ocn_out_env ocn.log
setenv atm_out_env atm.log
setenv land_out_env land.log
setenv cpl_out_env cpl.log

poe
```

Contents of file "tasklist":

```
ice
ice
ocn
ocn
ocn
ocn
land
land
atm
atm
atm
atm
cpl
cpl
```

Algorithms and Implementation

- Why do we call initial setup process “component handshaking”, instead of “executable handshaking”?
- Create unique MPI communicator for each component: local_comp_world
- Trivial overhead

Single-Component Executable Handshaking

- Root proc reads registration file, then broadcast
- Every proc knows total # of exes, and is assigned a unique exe_id
- Use exe_id as color, call MPI_comm_split to create local exe_world
- Local comp_world = local exe_world

Multi-Component Executable Handshaking

- Use unique exe_id as color, call MPI_comm_split to create local exe_world
- Components non-overlapping
  - each comp has unique comp_id
  - use comp_id as color to call MPI_comm_split
- Components overlapping
  - loop through all comps in each executable
  - set color=1 for this comp, color=0 for others
  - Repeatedly call MPI_comm_split, creating one local communicator for one comp at a time
  - Order of total # of comps

Status

- Completed MPH1, MPH2, MPH3, MPH4
  - Software available free online: http://hpcrd.lbl.gov/SCG/acpi/MPH
  - Complete users manual
- MPH runs on
  - IBM SP
  - SGI Origin
  - HP Compaq clusters
  - PC Linux clusters

MPH Users

- MPH users
  - NCAR CCSM
  - CSU geodesic grid coupled climate model
  - NCAR/WRF, for coupled models
- People expressed clear interests in using MPH
  - SGI/NASA, Irene Carpenter / Jim Taft, on SGI for coupled models
  - UK ECMWF, for ensemble simulations
  - Germany, Johannes Diemer, for coupled model on HP clusters
  - NOAA, for coupling models over grids

Future Work

- Flexible way to handle SMP nodes for MPI tasks
- Dynamic component model processor allocation or migration
- Extension to do model integration over grids
- A C/C++ version
- Multi-instance runs for multi-component, multi-executable applications
- Single-executable CCSM development

Related Work

- Software industry
  - Visual Basic, CORBA, COM, Enterprise JavaBeans
- HPC: Common Component Architecture (CCA)
  - CCAFFEINE, Unitah, GrACE, CCAT, XCAT
- Domain-specific Frameworks
  - Earth System Model Framework (ESMF)
  - PETSc, POOMA, Overture, Hypre, CACTUS
- Problem Solving Environment (PSE)
  - Purdue PSEs, ASCI PSE, Jaco3, JULIUS, NWChem

Summary

- Multi-Component Approach for large & complex application software
- MPH glues together distributed components
- Main Functionality:
  - flexible component name registration
  - run-time resource allocation
  - inter-component communication
  - query multi-component environment
- Five Execution Modes: SCSE, SCME, MCSE, MCME, MIME
- Easily switch between different modes

Status of Single-Executable CCSM Development

First Step

- Re-designed top level CCSM structure.
- Initial version completed (perform essential functions of Tony Craig's test code).
- All tested functions reproduced bit-to-bit agreement on NERSC IBM SP.

Resolved Issues (1)

- Co-existing with multi-executable code
- Flexible switching among different model options: real model, data model, dead (mock) model

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()
```

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 = real)   call cam2()

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

Resolved Issues (2)

- Allow MPI_tasks_per_node set differently on different components.
  - Schematically resolved (using task geometry and MPMD command file). Tested on IBM
  - Writing convenient way to specify this using MPH
- Allow OpenMP-threads set to different number on different components
  - Easily done for multi-executable
  - For single-exec, set from each component dynamically at runtime (instead of environmental variables). Tested on IBM

OpenMP_threads

- Multi-exec: specified as environment variable
- Single-exec: need to be model dependent, dynamically adjustable variables:

```
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
```

Resolved Issues (3)

- Resolved name conflict issue
- Propose module-based approach

Name Conflict in Single-Exec CCSM

- Different component models have subroutines with same name but different contents.
- Each subroutine name becomes a global symbol name
- Compiler generates a warning for multiple matches and always uses the 1st match

Two Probable Solutions

- One solution: rename in source codes
  - Renaming all functions, subroutines, interfaces, variables by adding a prefix
    - Substantial rework
- A module-based approach:
  - Key idea: Localization of global symbols
  - Using wrapper module with “include”
  - “Use Module Only” renaming
    - Minimal renaming
    - Only when different component modules appear in same file
  - less-tedious solution

Example

ocn_main.F atm_main.F

ocn1_mod.F atm1_mod.F

xyz2.F ⟵ conflict ⟶ xyz2.F

================================

ocn_wrapper.F:

```
        module ocn_wrapper
        use ocn1_mod
        contains
# include "xyz1.F"
# include "xyz2.F"   ! Local symbol
# include "xyz3.F"
        end module
```

================================

ocn_main.F: use ocn_wrapper

Public Variables, Functions, Interfaces

They are still global symbols and cause conflicts between component models.

Renaming conflict names on the fly:

Suppose 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
```

This also works for variables and interfaces.
Concrete examples see http://hpcrd.lbl.gov/SCG/acpi/SE

Immediate Plan

- Implement module-based approach for solving naming conflict in single-exec CCSM for data models and real models on IBM SP.
- Implement module-based approach in single-exec CCSM on other architectures.

Acknowledgement

- Collaborators
  - NCAR: Tony Craig, Brian Kauffman,
    Vince Wayland, Tom Bettge
  - Argonne National Lab: Rob Jacobs, Jay Larson
- Resources
  - DOE SciDAC Climate Project
  - NERSC Program