Title

Permalink
https://escholarship.org/uc/item/5t56c7js

Authors
He, Yun (Helen)
Ding, Chris H.Q.

Publication Date
2002-06-20
Multi Program-Components Handshaking (MPH) Utility

Yun He and Chris Ding
NERSC Division, Lawrence Berkeley National Laboratory

June 20, 2002

Contents

1 Introduction 3

2 How to Use 5

3 How to Compile and Run 5

4 Acknowledgement 7

5 Routine/Function Prologues 8

5.1 Module MPH_module – Multi Program-Components Handshaking (Source File: mph.F) 8

5.1.1 MPH_components – main MPH setup function 8

5.1.2 MPH_init – initialize MPI and read the processors map info 10

5.1.3 MPH_local – local handshaking 12

5.1.4 MPH_global – global handshaking 12

5.1.5 PE_in_component – check if a processor is in a component 13

5.1.6 PE_in_num_comps – return the number of components a processor 14

5.1.7 MPH_global_id – find global processor id 15

5.1.8 MPH_comm_join – join two components 15

5.1.9 MPH_redirect_output – redirect output from each component 16

5.1.10 MPH_read_list – read and process info from "processors_map.in" 17

5.1.11 MPH_find_name – find name in a namelist 18

5.1.12 MPH_help – display help info 18

5.1.13 MPH_debug – define debug level 20

5.1.14 MPH_timer – collect timing info in different channels 20

5.1.15 MPH_total_components – find number of total components 21

5.1.16 MPH_comp_name – find component name given component id 22

5.1.17 MPH_comp_id – find component id given component name 23

5.1.18 MPH_local_world – find local communicator given component name 24

5.1.19 MPH_exe_id – find executable id given component name 24

5.1.20 MPH_total_num_exe – find total number of executables 25
5.1.21 MPH_num_comps – find number of components in an executable . . 26  
5.1.22 MPH_local_proc_id – find local processor id in a component . . . . 26  
5.1.23 MPH_local_totProcs – find total number of processors . . . . . . . . 27  
5.1.24 MPH_global_proc_id – find global processor id . . . . . . . . . . . . 28  
5.1.25 MPH_global_proc_id – find total number of processors . . . . . . . . 29  
5.1.26 MPH_local_world – find local communicator of an executable . . . . 29  
5.1.27 MPH_exe_low_proc_limit - find lower processor limit of a component 30  
5.1.28 MPH_exe_up_proc_limit - find upper processor limit of a component 31
1 Introduction

MPH version 2 combines all features of MPH version 1, unifies the interfaces, and provides more flexible components integration/execution modes.

In a distributed multi-component environment, each executable resides on a set of SMP nodes. Components within an executable may overlap on different nodes or processors.

MPH Version 2 contains the following functionality:

- component name registration
- resource allocation
- multi-component single executable, multi-component multi-executable, etc.
- inter-component communication
- inquiry on the multi-component environment
- standard in/out redirect

Please see more information at
http://www.nersc.gov/research/SCG/acpi/MPH
and please list the following in your reference if useful:
"MPH: a Library for Distributed Multi-Component Environment"

Consider the entire simulation system (CCSM) consists of many executables, each executable containing one or more components. This architecture offers complete flexibility, and is consistent with CORBA, DCE, CCA et al.

1) Every executable starts with

```c
mpi_exec_world = &
    MPH_components(name1='ocean', name2='atmosphere',...)
```

You may have only one component in this executable, or up to 10 components in this executable. Component names are nametags (place holder) and are completely arbitrary. They must be self-consistently used, and match the "processors_map.in" registration file. This setup subroutine replaces MPH_setup() in MPH version 1. All other MPH functionality remains identical.

2) Some usages:
a) CCSM example. Ice & Land share one executable.

   coupler - one executable
   atmosphere - one executable
   ocean - one executable
   ice & land - one executable with 2 components,
                 they may overlap on processors

b) CCSM example. Multiple instances of atmosphere.

   coupler - one executable
   atmosphere - one executable of 3 components
                 each is a CCM instance of a different Dycore.
   ocean - one executable
   land - one executable with 3 components for CCMs.
          each is a land model to match the CCM
   ice - one executable

c) PCM example.

   couple - one executable
   atmosphere & land - one executable
   ocean & ice - one executable

3) "processors_map.in" registration file

The following example contains 3 executables:
   1st executable has a single component: coupler
   2nd executable has 2 components: ocean, ice
   3rd executable has 3 components: atmosphere, land, chemistry

BEGIN
   coupler
   Multi_Comp_Start
   2
   ocean  0  3
   ice    4 10
   Multi_Comp_End
   Multi_Comp_Start
   3
   atmosphere 0 10
   land    11 13
   chemistry 14 25
   Multi_Comp_End
END

a) Allocation of processors for each executable is controlled by job launching process (different on IBM, SGI, Compaq).

b) Processor ranges for each components are defined local to the executable.

2 How to Use

Users need to "use MPH_module" in the application codes, and invoke the appropriate MPH_components function for the multiple components in each executable. For example, ICE_LAND_World = MPH_components (name1="ice", name2="land"). You could use MPH_debug call to determine the output message amount, the default level is 0. "MPH_help" call provides you the available inquiry functions for that mode.

An input file called "processors_map.in" to give detailed information of component nametags and processor ranges. See more detail about how this file looks like in Section 1.

Each component maintains its own output in a separate file (file name defined by environment variable either in command line or in batch run script), assuming the local processor 0 of each component being responsible for most output, other occasional writes from all the components are stored in one combined standard output file.

This is accomplished by processor rank 0 of each component call subroutine "MPH_redirect_output" with the model name as argument. IBM and SGI could do the output redirect with the help of system function "getenv" or "pxfgetenv". Compaq cannot do this. And T3E is able to get the correct output files created using "pxfgetenv", but only output with those "write(6,*)" could be redirected, but not those with "write(*,*)", since * is equal to unit 101, and permanently related to the non-redirectable stdout.

3 How to Compile and Run

The shared "Makefile" detects the machine architecture and
compiles appropriately for IBM, SGI and Compaq. For test case 1, type "make test1", and for test case 2, type "make test2". or "gmake ..." depends on your machine).

After compile, you will have executables generated ("ice_land", "cpl", "pop_atm" for test1, and "ice_land", "cpl" for test2) in the corresponding subdirectory. Each sample subdirectory also includes batch scripts and sample output.

Go to that directory first (here we use test2 as an example), and then:

1) To run on NERSC and NCAR IBM SP interactively:
   a) % unsetenv MP_TASKS_PER_NODE
   b) % setenv ice_out_env ice.log
      % setenv land_out_env land.log
      % setenv cpl_out_env cpl.log
   c) Make sure the following command in ONE LINE:
      % poe -pgmmodel mpmd -cmdfile tasklist -nodes 3 -procs 6
         -stdoutmode ordered -infolevel 2 > & output &

This is to run the executables listed in user supplied "tasklist" in the mpmd mode on total of 3 nodes and 6 procs.

And "tasklist" looks like this:
   ice_land
   ice_land
   cpl
   cpl
   ice_land
   ice_land

To run on IBM SP with batch script:
% llsubmit runscript.ibm

And "runscript.ibm" looks like this:
   #!/usr/bin/csh -f
   # @ output = poe.stdout.$(jobid).$.stepid
   # @ error = poe.stderr.$(jobid).$.stepid
   # @ class = debug
   # @ job_type = parallel
   # @ task.geometry = {(0,2)(1,3)(4,5)}
   # @ total_tasks=6
   # @ network.MPI = css0, not_shared, us
   # @ queue
setenv MP_PGMODEL   mpmd
setenv MP_CMDFILE   tasklist
setenv ice_out_env  ice.log
setenv land_out_env land.log
setenv cpl_out_env  cpl.log
poe

Again, it needs a user supplied "tasklist", and it runs in mpmd mode.
The task.geometry keyword specifies which tasks run in the same node.

2) We could not run it on NERSC CRAY T3E since there is no mpmd
mechnism.

3) To run on NCAR SGI interactively:
   a) % setenv ice_out_env ice.log
       % setenv land_out_env land.log
       % setenv cpl_out_env cpl.log
   b) % mpirun -p "[%g]" -np 4 ice_land : -np 2 cpl > output

This is to run ice_land on 4 procs and cpl on 2 procs.
[%.g] is to print the global id as a prefix for each output line.

4) To run on NCAR Compaq with batch script:
   % prun -n6 -t runscript.dec

   And "runscript.dec" looks like this:
   
   #! /bin/csh
   if ($RMS_RANK >= 0 && $RMS_RANK <= 3) ice-land &
   if ($RMS_RANK >= 4 && $RMS_RANK <= 5) cpl &
   exit

4  Acknowledgement

MPH is developed in collaboration with Tony Craig, Brian Kauffman, Vince Wayland and
Tom Bettge of National Center of Atmospheric Research, and Rob Jacobs and Jay Larson
of Argonne National Laborotory. Vince Wayland of NCAR contributes for Makefiles on SGI
and Compaq. Dan Anderson and Bill Celmaster of NCAR also help in the batch run script
of Compaq. This work is supported by the Office of Biological and Environmental Research,
Climate Change Prediction Program, under ACPI Avant Garde project, and by the Office
of Computational and Technology Research, Division of Mathematical, Information, and
Computational Sciences, both of the U.S. Department of Energy under contract number
DE-AC03-76SF00098.
5 Routine/Function Prologues

5.1 Module MPH_module – Multi Program-Components Handshaking
(Source File: mph.F)

This module multiple executables with multiple components in each executable. This mod-
ule multiple executables with multiple components in.

REVISION HISTORY:

2001-Nov-15 -- add PROTEX convention
2001-May-20 -- first prototype

INTERFACE:

module MPH_module

USES:

implicit none
include 'mpif.h'
private ! except

PUBLIC MEMBER FUNCTIONS:

classic :: MPH_components
classic :: PE_in_component
classic :: PE_in_num_comps
classic :: MPH_global_id
classic :: MPH_comm_join
classic :: MPH_redirect_output
classic :: MPH_help
classic :: MPH_debug
classic :: MPH_timer
classic :: MPH_total_components
classic :: MPH_comp_name
classic :: MPH_comp_id
classic :: MPH_local_world
classic :: MPH_exe_id
classic :: MPH_total_num_exe
classic :: MPH_num_comps
classic :: MPH_local_proc_id
classic :: MPH_local_totProcs
classic :: MPH_global_proc_id
classic :: MPH_global_totProcs
classic :: MPH_exe_world
public :: MPH_exe_low_proc_limit
public :: MPH_exe_up_proc_limit

PUBLIC DATA MEMBERS:

integer, public :: istatus(MPI_STATUS_SIZE), ierr
integer, public :: MPH_Global_World ! total processor for the whole world

DEFINED PARAMETERS:

integer, parameter :: max_num_comps=10 ! maximum number of components
integer, parameter :: maxProcs_comp=128 ! maximum number of procs per comp
integer, parameter :: max_num_exes=10 ! maximum number of executables
integer, parameter :: N_CHANNELS=10 ! number of channels for timing

LOCAL VARIABLES:

type Acomponent
  character (len=32) :: name ! component name
  integer :: num_process ! number of processors
  integer :: process_list (maxProcs_comp)
                 ! global processor_id, increasing order
end type Acomponent

type (Acomponent) :: components (max_num_comps) ! allocate components
integer :: MPI_Acomponent

integer :: local_world (max_num_comps) ! communicator for each component
integer :: local_proc_id (max_num_comps) ! proc id in each component
integer :: local_totProcs (max_num_comps)
             ! total number of processors in each component
integer :: global_proc_id ! proc id in the whole world
integer :: global_totProcs ! total number of processors

integer :: COMM_master ! communicator for submaster of each component
integer :: total_components ! total number of components
character (len=32) :: component_names (max_num_comps) ! component names
character (len=32) :: name (max_num_comps) ! name array used in setup
integer comp_id (max_num_comps) ! component id of each component

integer :: num_comps (max_num_comps)
        ! number of components in each executable
integer :: exe_low_proc_limit (max_num_comps)
        ! lower processor limit of each component
        ! in each executable world
integer :: exe_up_proc_limit (max_num_comps)
        ! upper processor limit of each component
        ! in each executable world
integer :: exe_world_proc_id (max_num_exes)
        ! processor id in the executable world
integer :: exe_world_totProcs (max_num_exes)
        ! number of processors in each executable
integer :: exe_world (max_num_exes)
        ! communicator for each executable
integer :: exe_ids (max_num_comps) ! executable ids
integer :: total_num_exe    ! total number of executables
integer :: exe_id          ! executable id

integer :: debug_level = 0    ! level of debug
.. for timer ..
real (kind=8) :: init_time = -1.0
real (kind=8) :: last_time, tot_time (0:N_CHANNELS)

5.1.1 MPH_components – main MPH setup function

This is the main function for each of the executable to call to setup the distributed multi-component environment. For example, if ocean and atmosphere sits in one executable, the source code will contain:

    mpi_exec_world =
        MPH_components(name1='ocean', name2='atmosphere',...)

This function returns the MPI communicator of local executable world.

REVISION HISTORY:

2001-Dec-03 -- reduce from 10 arguments to 5
2001-Nov-15 -- add PROTEX convention
2001-May-20 -- first prototype

INTERFACE:

    integer function MPH_components(name1, name2, name3, name4, name5)

USES:

    implicit none
INPUT PARAMETERS:

These are component names
character(len=*) , intent(in) :: name1
character(len=*) , intent(in) , optional :: name2 , name3 , name4 , name5

OUTPUT PARAMETERS:

! This function returns the MPI communicator of this executable
! world: exe_world (exe_id)

SEE ALSO:

MPH_init , MPH_local , MPH_global , MPH_debug

LOCAL VARIABLES:

integer :: k

5.1.2 MPH_init – initialize MPI and read the processors map info

This routine calls mpi_init , obtains global processor id. It reads and processes the “processors_map.in” file. It also defines an MPI_Acomponent structure (includes component name, number of processors and processor list for each component) for easy gather and scatter.

REVISION HISTORY:

2001-Nov-15 -- add PROTEK convention
2001-May-20 -- first prototype

INTERFACE:

subroutine MPH_init ()

USES:

implicit none

SEE ALSO:

MPH_read_list , MPH_local , MPH_global

LOCAL VARIABLES:

integer :: iblock(3) , idisp(3) , itype(3)
5.1.3 MPH_local – local handshaking

This routine first defines exe_jd, and creates local exe_world for each executable. It then gathers global processor ids onto submaster (whose rank is 0 in exe_world). And then it creates local_world for each component within exe_world based on its upper and lower processor limits. Finally it collects name, number of processors, and processor list of each component onto submaster of each executable world.

REVISION HISTORY:

2001-Dec-13 -- add warning for overlapping processors
2001-Nov-27 -- add local_totProcs for single component executables
2001-Nov-19 -- add PROTEX convention, use new MPH_read_list interface
2001-May-20 -- first prototype

INTERFACE:

subroutine MPH_local ()

USES:

implicit none

SEE ALSO:

MPH_init, MPH_global, MPH_find_name

LOCAL VARIABLES:

integer :: color, key
integer :: id, comp_id_end, i, k

5.1.4 MPH_global – global handshaking

This routine first creates an MPI communicator COMM_master for all submasters (whose rank is 0 in the executable world). It then does an MPI_allgatherv in COMM_master to collect all the components information from each submaster. Then each submaster broadcasts AComponents to all PEs in its local exe_world. Finally, every processor lists the complete info of all the components.

REVISION HISTORY:
2002-Apr-15 -- correct a bug in declaring sendbuf
2001-Dec-13 -- add warning for overlapping processors
2001-Nov-19 -- add PROTEX convention
2001-May-20 -- first prototype

INTERFACE:

subroutine MPH_global ()

USES:

implicit none

SEE ALSO:

MPH_init, MPH_local, MPH_debug

LOCAL VARIABLES:

integer :: id, i, color, key

type (Acomponent) :: sendbuf(max_num_comps)

integer :: sendcount

integer :: recvcounts(0:total_num_exe-1)

integer :: displs(0:total_num_exe-1)

5.1.5 PE_in_component – check if a processor is in a component

This is a logical function to check if a processor is in a component.

REVISION HISTORY:

2001-Nov-15 -- add PROTEX convention
2001-May-20 -- first prototype

INTERFACE:

logical function PE_in_component (name, comm)

USES:

implicit none
SEE ALSO:

MPH_find_name

INPUT PARAMETERS:

character(len=*) , intent(in) :: name     ! component name

OUTPUT PARAMETERS:

! the local communicator of that component is written in comm.
integer , intent(out) :: comm     ! communicator for the component

LOCAL VARIABLES:

integer :: id , i

5.1.6 PE_in_num_comps – return the number of components a processor
in
This function returns the number of components a processor is in.

REVISION HISTORY:

2001-Dec-13 -- first prototype

INTERFACE:

integer function PE_in_num_comps ()

USES:

implicit none

SEE ALSO:

PE_in_component

LOCAL VARIABLES:

integer :: id , i
5.1.7 MPH_global_id – find global processor id

This function returns global processor id given the component name and local processor id in that component.

REVISION HISTORY:

2001-Nov-15 -- add PROTEX convention
2001-May-20 -- first prototype

INTERFACE:

integer function MPH_global_id (cname, lid)

USES:

implicit none

SEE ALSO:

MPH_find_name

INPUT PARAMETERS:

character(len=*), intent(in) :: cname ! component name
integer, intent(in) :: lid

! local processor id in the component

OUTPUT PARAMETERS:

! This function returns global_proc_id given the component
! name and local_proc_id in that component.

LOCAL VARIABLES:

integer :: temp

5.1.8 MPH_comm_join – join two components

This routine creates a joined MPI communicators for any two components. The order of these two components appeared in the subroutine parameter argument has an effect on the local process id within the joined communicator.

REVISION HISTORY:
2001-Nov-15 -- add PROTEX convention
2001-May-20 -- first prototype

INTERFACE:

subroutine MPH.comm_join (name1, name2, comm Joined)

USES:

implicit none

SEE ALSO:

MPH_find_name, PE_in_component

INPUT PARAMETERS:

character(len=*) , intent(in) :: name1, name2 ! two component names

OUTPUT PARAMETERS:

integer, intent(out) :: comm Joined
                      ! joined communicator for two components

LOCAL VARIABLES:

integer :: id1, id2
integer :: color, key
logical :: con1, con2
integer :: comm1, comm2

5.1.9 MPH redirect output – redirect output from each component

This routine redirects output to a log file defined by an environment variable. System
functions ("getenv" for IBM and "pxfgetenv" for SGI and T3E) are used to retrieve the
environment variable.

REMARKS:
In order to redirect component output to a separate file, a user will setup something like the following in the run script:

```plaintext
setenv ice_out_env ice.log
setenv land_out_env land.log
setenv cpl_out_env cpl.log
```

**REVISION HISTORY:**

- 2001-Nov-15 -- add PROTEX convention
- 2001-May-20 -- first prototype

**INTERFACE:**

```plaintext
subroutine MPH_redirect_output (name)
```

**USES:**

```plaintext
implicit none
```

**INPUT PARAMETERS:**

```plaintext
character(len=*) , intent(in) :: name ! part of the log file name
```

**LOCAL VARIABLES:**

```plaintext
integer :: lenname, lenval, rcode
character(len=32) :: output_name_env
character(len=64) :: output_name, temp_value
```

---

**5.1.10 MPH_read_list – read and process info from "processors_map.in"**

This routine reads and processes info from "processors_map.in". Please see a sample input file in Introduction.

**REVISION HISTORY:**

- 2001-Nov-15 -- add PROTEX convention
  - remove two arguments: max_num_comp, max_num_exe
- 2001-May-20 -- first prototype

**INTERFACE:**
integer function MPH_read_list (filename, filetag, namelist, 
&
   low, up, local_num, id_exe, num_comp, total_exe)

USES:

implicit none

SEE ALSO:

MPH_find_name, MPH_init

INPUT PARAMETERS:

character(len=*) , intent(in) :: filename  ! the input file name
character(len=*) , intent(in) :: filetag   ! "PROCESSORS_MAP"

OUTPUT PARAMETERS:

character (len=32) , intent(out) :: namelist(max_num_comps)  
! component names
integer , intent(out) :: low(max_num_comps)  
! lower processor limit of each component  
! in each executable world
integer , intent(out) :: up(max_num_comps)  
! upper processor limit of each component  
! in each executable world
integer , intent(out) :: local_num(max_num_comps)  
! total number of processors for each component
integer , intent(out) :: id_exe(max_num_comps)  ! executable ids
integer , intent(out) :: num_comp(max_num_comps)  
! number of components in each executable
integer , intent(out) :: total_exe  ! total number of executables

LOCAL VARIABLES:

integer :: i, k, multi_num, id
character (len=32) :: frontline, temp
integer :: itemp1, itemp2
5.1.11 MPH\_find\_name – find name in a namelist

This routine finds if a certain name exists in an array of namelist and returns the rank if it does or -1 if it does not.

**REVISION HISTORY:**

2001-Nov-15 -- add PROTEX convention
2001-May-20 -- first prototype

**INTERFACE:**

integer function MPH\_find\_name (name, namelist, num)

**USES:**

implicit none

**INPUT PARAMETERS:**

character(len=\*), intent(in) :: name     ! name to be found
integer :: num                     ! length of namelist array
character (len=32) namelist(num)   ! name list array

**OUTPUT PARAMETERS:**

! the rank of a name in an array of namelist or -1 if not exist

**LOCAL VARIABLES:**

integer :: i

5.1.12 MPH\_help – display help info

This routine displays some help info for the MPH setup interface and some inquiry functions.

**REVISION HISTORY:**

2001-Nov-15 -- add PROTEX convention
2001-May-20 -- first prototype

**INTERFACE:**
subroutine MPH_help (arg)

USES:
implicit none

INPUT PARAMETERS:
character(len=*) , intent(in) :: arg  ! either 'on' or 'off'

5.1.13  MPH_debug -- define debug level

This routine defines the debug level. The higher the level is, the more debug information
the code will display.

REVISION HISTORY:
2001-Nov-15 -- add PROTEXT convention
2001-May-20 -- first prototype

INTERFACE:
subroutine MPH_debug (level)

USES:
implicit none

INPUT PARAMETERS:
integer, intent(in) :: level  ! 0 (default), 1 or 2

5.1.14  MPH_timer -- collect timing info in different channels.

This function collects timing info in different channels.

Usage:
channel 0 is the default channel, using init_time.
-----------------------------------------------
timer calls to walk-clock dclock(), and do the following:
flag=0 : Sets initial time; init all channels.  
flag =1 : Calculates the most recent time interval; accrues it to the specified channel (default 0); Returns it to calling process.  
flag =2 : Calculates the most recent time interval; accrues it to the specified channel (default 0); Returns the current total time in the specified channel.  

REVISION HISTORY:

2001-Nov-19 -- add PROTEK convention  
2001-May-20 -- first prototype

INTERFACE:

real (kind=8) function MPH_timer (flag, channel)

USES:

implicit none

INPUT PARAMETERS:

integer :: flag, channel

OUTPUT PARAMETERS:

real (kind=8) :: new_time, delta_time, MPI_Wtime

LOCAL VARIABLES:

real (kind=8) :: MPH_timer

5.1.15  MPH._total_components -- find number of total components
This function returns the number of total components.

REVISION HISTORY:
2001-Nov-15 -- add PROTEX convention
2001-May-20 -- first prototype

INTERFACE:

    integer function MPH_total_components ()

USES:

    implicit none

OUTPUT PARAMETERS:

    ! total_components

5.1.16  MPH_comp_name -- find component name given component id

This function returns component name given component id.

REVISION HISTORY:

    2001-Dec-13 -- use optional argument
    2001-Nov-15 -- add PROTEX convention
    2001-May-20 -- first prototype

INTERFACE:

    character (len=32) function MPH_comp_name (cid)

USES:

    implicit none

SEE ALSO:

    MPH_find_name, MPH_comp_id, MPH_comm

INPUT PARAMETERS:

    integer, intent(in), optional :: cid    ! component id
OUTPUT PARAMETERS:

! component_names (cid)

LOCAL VARIABLES:

  integer :: id, comm

5.1.17  MPH_comp_id – find component id given component name

This routine returns component id given component name.

REVISION HISTORY:

  2001-Dec-13 -- use optional argument
  2001-Nov-15 -- add PROTEK convention
  2001-May-20 -- first prototype

INTERFACE:

  integer function MPH_comp_id (cname)

USES:

  implicit none

SEE ALSO:

  MPH_find_name, MPH_comp_name, MPH_comm

INPUT PARAMETERS:

  character(len=*) , intent(in) , optional :: cname   ! component name

OUTPUT PARAMETERS:

  ! MPH_comp_id

LOCAL VARIABLES:

  integer :: id, comm
5.1.18 MPH_local_world – find local communicator given component name
This routine returns local communicator given component name.

REVISION HISTORY:
  2001-Dec-13 -- first prototype

INTERFACE:
    integer function MPH_local_world (cname)

USES:
    implicit none

SEE ALSO:
    MPH_find_name, MPH_comp_id

INPUT PARAMETERS:
    character(len=*_), intent(in), optional :: cname  ! component name

OUTPUT PARAMETERS:
    ! MPH_local_world

LOCAL VARIABLES:
    integer :: id, comm

5.1.19 MPH_exe_id – find executable id given component name
This function returns the executable id given component name.

REVISION HISTORY:
  2001-Dec-13 -- use optional argument
  2001-Nov-15 -- add PROTEX convention
  2001-May-20 -- first prototype
INTERFACE:
    integer function MPH_exe_id (cname)

USES:
    implicit none

SEE ALSO:
    MPH_find_name

INPUT PARAMETERS:
    character(len=*) , intent(in) , optional :: cname ! component name

OUTPUT PARAMETERS:
    ! exe_ids (id)

! LOCAL PARAMETERS:
    integer :: id , comm

5.1.20  MPH_total_num_exe -- find total number of executables
This function returns the total number of executables.

REVISION HISTORY:
    2001-Nov-15 -- add PROTEX convention
    2001-May-20 -- first prototype

INTERFACE:
    integer function MPH_total_num_exe ()

USES:
    implicit none

OUTPUT PARAMETERS:
    ! total_num_exe
5.1.21 **MPH_num_comps** – find number of components in an executable

This function returns number of components in an executable given the executable id.

**REVISION HISTORY:**

- 2001-Dec-13 -- use optional argument
- 2001-Nov-15 -- add PROTEK convention
- 2001-May-20 -- first prototype

**INTERFACE:**

```fortran
integer function MPH_num_comps (eid)
```

**USES:**

```fortran
implicit none
```

**INPUT PARAMETERS:**

- integer, intent(in), optional :: eid  ! executable id

**OUTPUT PARAMETERS:**

- ! num_comps (eid)

---

5.1.22 **MPH_local_proc_id** – find local processor id in a component

This function returns the local processor id given the component id.

**REVISION HISTORY:**

- 2001-Dec-13 -- use optional argument
- 2001-Nov-15 -- add PROTEK convention
- 2001-May-20 -- first prototype

**INTERFACE:**

```fortran
integer function MPH_local_proc_id (cid)
```

**USES:**

```fortran
implicit none
```
5.1.23 MPH_local_totProcs – find total number of processors

in a component.
This function returns the total number of processors in a component given the component id.

REVISED HISTORY:

2001-Dec-13 -- use optional argument
2001-Dec-13 -- use optional argument
2001-Nov-27 -- first prototype

INTERFACE:

integer function MPH_local_totProcs (cid)

USES:

implicit none

SEE ALSO:

MPH_global_totProcs

INPUT PARAMETERS:
integer, intent(in), optional :: cid ! component id

OUTPUT PARAMETERS:

! local_totProcs (cid)

LOCAL VARIABLES:

integer :: id, comm

5.1.24 MPH_global_proc_id – find global processor id

This function returns the global processor id.

REVISION HISTORY:

2001-Nov-15 -- add PROTEK convention
2001-May-20 -- first prototype

INTERFACE:

integer function MPH_global_proc_id ()

USES:

implicit none

SEE ALSO:

MPH_local_proc_id

OUTPUT PARAMETERS:

! global_proc_id
5.1.25  MPH_global_proc_id – find total number of processors

This function returns the total number of processors in MPH world.

REVISION HISTORY:

2001-Nov-15 -- add PROTEK convention
2001-May-20 -- first prototype

INTERFACE:

integer function MPH_global_totProcs ()

USES:

implicit none

OUTPUT PARAMETERS:

! global_totProcs

5.1.26  MPH_local_world – find local communicator of an executable

This function returns the local MPI communicator of an executable given the executable id.

REVISION HISTORY:

2001-Dec-13 -- change function name from MPH_local_world to MPH_exe_world,
             use optional argument
2001-Nov-15 -- add PROTEK convention
2001-May-20 -- first prototype

INTERFACE:

integer function MPH_exe_world (eid)

USES:

implicit none

INPUT PARAMETERS:
integer, intent(in), optional :: eid  ! executable id

OUTPUT PARAMETERS:

! exe_world (eid)

5.1.27  MPH_exe_low_proc_limit - find lower processor limit of a component
This function returns the relative lower processor limit of a component in the executable world.

REVISION HISTORY:

2002-Jun-20 -- correct the argument from eid to cid
2001-Dec-13 -- use optional argument
2001-Nov-15 -- add PROTEx convention
2001-May-20 -- first prototype

INTERFACE:

integer function MPH_exe_low_proc_limit (cid)

USES:

implicit none

SEE ALSO:

MPH_exe_up_proc_limit

INPUT PARAMETERS:

integer, intent(in), optional :: cid  ! component id

OUTPUT PARAMETERS:

! exe_low_proc_limit (cid)
5.1.28  MPH\texttt{\_exe\_up\_proc\_limit} - find upper processor limit of a component

This function returns the relative upper processor limit of a component in the executable world.

REVISED HISTORY:

2002-Jun-20 -- correct the argument from eid to cid
2001-Dec-13 -- use optional argument
2001-Nov-15 -- add PRUTEX convention
2001-May-20 -- first prototype

INTERFACE:

\begin{verbatim}
  integer function MPH\_exe\_up\_proc\_limit (cid)
\end{verbatim}

USES:

\begin{verbatim}
  implicit none
\end{verbatim}

SEE ALSO:

\begin{verbatim}
  MPH\_exe\_low\_proc\_limit
\end{verbatim}

\textbf{INPUT PARAMETERS:}

\begin{verbatim}
  integer, intent(in), optional :: cid  ! component id
\end{verbatim}

\textbf{OUTPUT PARAMETERS:}

\begin{verbatim}
  ! exe\_up\_proc\_limit (cid)
\end{verbatim}