A flexible instrumentation of ARPEGE/AROME

Philippe.Marguinaud@meteo.fr

1 Introduction

In this document, we describe a new mechanism added to gmkpack to make the dr_hook timing package usage more flexible. In particular, it is now possible to select from a namelist which Fortran units will be timed by dr_hook, on a file by file basis, or per project.

2 Usage

2.1 At compile time

In order to use this capability, ARPEGE/AROME must have been compiled with a special gmkpack option; compiling a pack relying on a base back with this option implies that the base pack has also been compiled with this special gmkpack option. If these assumptions are not fulfilled, the behavior is unspecified.

Enabling this feature in ARPEGE/AROME involves editing the script generated by gmkpack and setting the environment variable GMK_DR_HOOK_ALL to a non-empty value (please note that the string “0” **is** a non-empty value). Hence, the following line should be added at the top of the gmkpack generated script:

```
export GMK_DR_HOOK_ALL=1
```

gmkpack will then generate extra-code for timing instrumentation. A special library and module interfaces will be created in the src/local/.dr_hook directory.

2.2 At run time

If a file named namelist.hook_all exists, then the executable will open it and attempt to read the following namelist:

```
logical :: lhook
character(len=32) :: projects_hook(3*pmax)
character(len=64) :: files_hook(3*fmax)
namelist /namhook__all/ lhook, projects_hook, files_hook
```

The meaning of these parameters is:

- lhook : if set to true will enable dr_hook in the whole source code
- projects_hook : list of projects where dr_hook should be enabled or disabled. To disable (resp. enable) a project, it is necessary to prefix it with a '-' (resp. '+')
- files_hook : list of files where dr_hook should be enabled. To enable (resp. disable) dr_hook

in a file, it is necessary to prefix it with a '+' (resp '-').

These parameters are handled sequentially: first lhook is used as a background default value, then the project list, then the file list.

For instance:

```
& namhook__all
  projects_hook(1) = '+arp'
  projects_hook(2) = '+ald'
  files_hook(1) = '-arp/nmi/mo3dprjad.F90'
  files_hook(2) = '-arp/nmi/reordo3ad.F90'
/
```

Will enable dr_hook in the source of the arp and ald projects, except in the arp/nmi/mo3dprjad.F90 and arp/nmi/reordo3ad.F90 files.

3 Design and inner workings

The dr_hook library provides the user with a single boolean named “lhook”; upon this single variable depends the activation of dr_hook measurements on the whole code. Enabling instrumentation on a file by file basis involves having a boolean flag per file.

The drhook_all.pl script scans the source code and generates modules with a boolean flag per file unit; when gmkpack compiles the Fortran source code, extra-options are provided to the compiler and linker so that these booleans be used in place of the lhook flag of the yomhook module. This is the basic design that we detail below.

3.1 Interfacing with gmkpack

3.1.1 Instrumentation library creation

This library is created by the drhook_all.pl script:

```
$ drhook_all.pl --create-library --fc="$ICS_F90_CPP" --ar="$AR -qv"
```

This command is issued in the icspack.sh utility.

The library is created in the src/local/.dr_hook directory.

This library relies on dr_hook, hence parkind1.F90 and yomhook.F90 are copied from the main source directory and pre-compiled here before doing anything else.

The script drhook_all.pl has a “--debug” option which triggers the generation of extra debugging code. It is possible to enable drhook_all.pl debugging by setting the following environment variable:

```
$ export GMK_DR_HOOK_ALL_FLAGS=”--debug”
```

3.1.2 Compiler options

Compiler options are provided by the drhook_all.pl script:

```
$ drhook_all.pl --fflags --full-cpp-flags arp/programs/master.F90
-D_YOMHOOK__ALL -I /home/marguina/pack/drhook/src/local/.dr_hook
-Dyomhook=yomhook_000002 -DYOMHOOK=yomhook_000002 -Dlhook=lhook_f_0000c5
-DLHOOK=lhook_f_0000c5
```

From the output above, we see that arp/programs/master.F90 will be compiled with the following

options:

- -D_YOMHOOK__ALL may be used to enable a call to the setup of the yomhook__all library
- -I /home/marguina/pack/drhook/src/local/.dr_hook makes the generated modules visible
- -Dyomhook=yomhook_000002 and -DYOMHOOK=yomhook_000002 makes the file being compiled use the yomhook_000002 module in place of the traditional yomhook module.
- -Dlhook=lhook_f_0000c5 and -DLHOOK=lhook_f_0000c5 makes the compiler load the lhook flags from the generated modules rather from the original yomhook module.

The preprocessor options trick does not work on files without the proper extension; hence it is necessary to preprocess these files with the drhook_all.pl script before compiling them:

```
$ drhook_all.pl --base-dir=$MKTOP/$branch –preprocess
```

Compiler options are set up in the gmkpack utility FORTRAN_cplpack.sh.

3.1.3 Linker options

Linker options are provided by the following command:

```
$ drhook_all.pl  --ldflags
-L /home/marguina/pack/drhook/src/local/.dr_hook -lyomhook__all
```

These options bring in the yomhook__all library.

Linker options are set up in the gmkpack utility loadpack.sh.

3.1.4 Cleaning

Cleaning the generated code is possible using the following command:

```
$ ./drhook_all.pl  --clean-code
```

Cleaning the databases is possible too:

```
$ ./drhook_all.pl  --clean
```

3.2 Anatomy of the yomhook__all library

The src/local/.dr_hook directory has the following contents:

```
$ ls *.f90
yomhook_000001.f90     yomhook_000025.f90  yomhook_000049.f90
yomhook_000002.f90     yomhook_000026.f90  yomhook_000050.f90
yomhook_000003.f90     yomhook_000027.f90  yomhook_000051.f90
...
yomhook_000022.f90     yomhook_000046.f90  yomhook_000070.f90
yomhook_000023.f90     yomhook_000047.f90  yomhook_000071.f90
yomhook_000024.f90     yomhook_000048.f90
yomhook__all.f90
```

The yomhook__all module contains code to initialize the yomhook__all library; it contains a single routine suhook__all, and a wrapper of the original dr_hook generic interface.

yomhook__all initialization is a three step process:

- initialize variables
- read the namelist from the file “namelist.hook_all” if it exists, otherwise return from the subroutine
- update the lhook individual flags according to the contents of the namelist read in the previous step

The code and data is distributed in several Fortran units, 100 ARPEGE/AROME source code units being handled in each, for the following reasons:

- having a single module holding the whole code and data is unacceptable; it takes a very long time to compile the code on some platforms
- having a module per ARPEGE/AROME source code unit is not possible, because the compiler would have to read/write several thousands of module interface files

Note that some of the choices made for the design of the yomhook__all library may look weird at first glance, but it is necessary to remind the reader that this code has been designed under compactness and performances constraints.

The file yomhook_000001.f90 handles the lhook flags of 100 ARPEGE/AROME source code files:

```
module yomhook_000001

use yomhook, only : dr_hook

logical, save :: lhook_f_000001 = .true. ! arp/phys_radi/srtm_taumol19.F90
logical, save :: lhook_f_000002 = .true. !
sur/module/surfexcdriverstl_ctl_mod.F90
logical, save :: lhook_f_000003 = .true. ! obt/bias_sat/calc_scan_1c.F90
logical, save :: lhook_f_000004 = .true. ! surfex/aux/get_sson.f90
...
end module

subroutine subhook1_000001(lhook)

use yomhook_000001

implicit none
logical, intent(in) :: lhook

lhook_f_000001 = lhook ! arp/phys_radi/srtm_taumol19.F90
lhook_f_000002 = lhook ! sur/module/surfexcdriverstl_ctl_mod.F90
lhook_f_000003 = lhook ! obt/bias_sat/calc_scan_1c.F90
lhook_f_000004 = lhook ! surfex/aux/get_sson.f90
lhook_f_000005 = lhook ! arp/module/yomoerr.F90
...
lhook_f_000064 = lhook ! sur/module/srft_mod.F90
end subroutine

subroutine subhook2_000001(px,nx)
use yomhook_000001
implicit none
integer :: nx
character(len=*) :: px(nx)

integer :: i
character :: c
```

```
do i = 1, nx
c = px(i)(1:1)
select case (px(i)(2:))
case ('odb/pandor/module')
select case (c)
case ('+')
lhook_f_000012 = .true.
case ('-')
lhook_f_000012 = .false.
end select
case ('mpa/chem/internals')
select case (c)
case ('+')
lhook_f_000019 = .true.
...
end subroutine

subroutine subhook3_000001(fx,nx)
use yomhook_000001
implicit none
integer :: nx
character(len=*) :: fx(nx)

integer :: i
character :: c

do i = 1, nx
c = fx(i)(1:1)
select case (fx(i)(2:))
case ('arp/phys_radi/srtm_taumol19.F90')
select case (c)
case ('+')
lhook_f_000001 = .true.
case ('-')
lhook_f_000001 = .false.
end select
case ('sur/module/surfexcdriverstl_ctl_mod.F90')
select case (c)
case ('+')
lhook_f_000002 = .true.
case ('-')
lhook_f_000002 = .false.
end select
case ('surfex/aux/get_sson.f90')
select case (c)
case ('+')
lhook_f_000003 = .true.
case ('-')
lhook_f_000003 = .false.
end select
...

end subroutine
```

From the code listed above, we see that the yomhook_000001 contains the flags for 100 of the ARPEGE/AROME files, that subhook1_000001 is responsible for global initialization of the individual hook flags, subhook1_000002 handles the per-project set-up of lhook_f flags, subhook1_000003 handles the per file set-up of lhook_f flags.

We are now ready to look at the yomhook__all source code:

```
module yomhook__all

use yomhook, only : yomhook_dr_hook => dr_hook
use parkind1, only : jpim, jprb

implicit none

public :: dr_hook

interface dr_hook
module procedure &
  dr_hook_default, &
  dr_hook_file, &
  dr_hook_size, &
  dr_hook_file_size, &
  dr_hook_multi_default, &
  dr_hook_multi_file, &
  dr_hook_multi_size, &
  dr_hook_multi_file_size
end interface

logical, save :: first_time = .true.

contains

subroutine dr_hook_default(cdname,kswitch,pkey)
character(len=*),   intent(in)    :: cdname
integer(kind=jpim), intent(in)    :: kswitch
real(kind=jprb),    intent(inout) :: pkey

call suhook__all
call yomhook_dr_hook(cdname,kswitch,pkey)

end subroutine dr_hook_default

...

subroutine suhook__all

integer, parameter :: pmax = 192
integer, parameter :: fmax = 7060

logical :: lhook

character(len=32) :: projects_hook(3*pmax)
character(len=64) :: files_hook(3*fmax)

namelist /namhook__all/ lhook, projects_hook, files_hook

integer :: ioerr

if(first_time) then
  first_time = .false.
else
  return
endif

lhook         = .false.
projects_hook = ''
files_hook    = ''

call subhook1_000001(lhook)
```

```
call subhook1_000002(lhook)
call subhook1_000003(lhook)
call subhook1_000004(lhook)
...
call subhook1_000070(lhook)
call subhook1_000071(lhook)

open (77, file = 'namelist.hook_all', form = 'formatted', status = 'old', iostat
= ioerr)
if (ioerr .ne. 0) return

read(77, nml = namhook__all, iostat = ioerr)
if (ioerr .ne. 0) call abor1 ('ERROR WHILE READING namhook__all')

close (77)

call subhook1_000001(lhook)
call subhook1_000002(lhook)
call subhook1_000003(lhook)
...
call subhook1_000069(lhook)
call subhook1_000070(lhook)
call subhook1_000071(lhook)
call subhook2_000001(projects_hook,size(projects_hook))
call subhook2_000002(projects_hook,size(projects_hook))
call subhook2_000003(projects_hook,size(projects_hook))
...
call subhook2_000069(projects_hook,size(projects_hook))
call subhook2_000070(projects_hook,size(projects_hook))
call subhook2_000071(projects_hook,size(projects_hook))

call subhook3_000001(files_hook,size(files_hook))
call subhook3_000002(files_hook,size(files_hook))
call subhook3_000003(files_hook,size(files_hook))
...
call subhook3_000069(files_hook,size(files_hook))
call subhook3_000070(files_hook,size(files_hook))
call subhook3_000071(files_hook,size(files_hook))

end subroutine
end module
```

All individual lhook flags are initialized to true; this implies that the first call to dr_hook encountered will initialize the yomhook__all library. Subsequent calls will not do that again, because of the first_time flag; the underlying assumption is that the first call to dr_hook will be issued from a non-multithreaded part of the code (such as the main program), otherwise, the code would require some protection in order to be thread-safe.

Initialization on first call is also the mechanism used in the original dr_hook library; this is why it was required to wrap calls to dr_hook.

3.3 Persistance of the yomhook__all flags

In order to be able to provide gmkpack with the correct flags (see the section compiler flags above), it is necessary to keep track of the following data:

- the associations ARPEGE/AROME files => yomhook module number.

  (eg arp/phys_radi/srtm_taumol19.F90 module number is yomhook_000001 in the code sample listed above)

  these data are stored in the dbm.pl file

- the associations ARPEGE/AROME files => lhook number.

  (eg sur/module/surfexcdriverstl_ctl_mod.F90 lhook number is 2 in the code sample above)

  these data are stored in the dbf.pl file
- the project ranking

  these data are stored in the dbp.pl file

Keeping these data is also necessary so that all yomhook__all code could be updated without being recompiled from scratch each time a new file is added to a pack. Only the latest created yomhook_NNNNNN.f90 and yomhook__all.f90 need to be compiled again.

Note also that once a file name has been added to the databases, it will never disappear, even if the ARPEGE/AROME file is physically removed. Removing it from the database would imply regenerating the whole set of yomhook_NNNNNN.f90 and recompiling them.

4 Planned changes

To make instrumentation easier, it would be desirable to replace the dr_hook code by directives.

```
USE YOMHOOK, ONLY : DR_HOOK, LHOOK
REAL(KIND=JPRB) :: ZHOOK_HANDLE
IF(LHOOK) CALL DR_HOOK('...',0,ZHOOK_HANDLE)
IF(LHOOK) CALL DR_HOOK('...',1,ZHOOK_HANDLE)
```

would become:

```
!USE_YOMHOOK
!DECLARE_ZHOOK
!DR_HOOK_ENTER
!DR_HOOK_QUIT
```

A preprocessor could then replace these directives with actual code before passing the code to the compiler.

This approach would make customized instrumentation much easier to implement. It should make it possible to select timed code sections not only by project or file name, but also on a per subroutine basis.