Dr.Hook – an instrumentation tool

by Sami Saarinen
Mats Hamrud, Deborah Salmond & John Hague
@ECMWF, Reading, UK
(June 15, 2005 FMI)

What is Dr.Hook ?

- A Fortran & C-callable instrumentation library to
  - Trap run-time problems
  - Gather performance profile info per subroutine
    - Wall-clock or CPU-times
    - Mflop/s & MIPS –rates (on some machines)
    - Memory usage profiling (on some machines)
    - Watchpoints for memory region(s) overwrites

- The basic feature: keep track of the calling tree
  - For every MPI-task and OpenMP-thread
  - Upon error (when caught via Unix-signals) tries to print the current active calling tree
  - The system specific traceback can also be printed

- System independent with low overhead
  - Basic < 1%, with MFlop/s counters ~ 1% (Power4)

What is Dr.Hook ? (cont'd)

- A traceback can also be printed at any time accompanied with memory, CPU, paging, wall-clock etc. info
- Run-time profile information
  - At exit prints gprof-like flat profile report for every instrumented routine per MPI-task
  - Each thread shown separately
  - Either wall-clock or CPU-time based
  - Mflop/s & MIPS-rates available

What is Dr.Hook ? (cont'd)

- □ Run-time memory profile information
  - ❍ On some machines (like IBM Power-series) we have intercepted Fortran90 ALLOCATE & DEALLOCATE (and all C-routines in ODB) with our own memory allocation routines to let Dr.Hook to keep track of memory usage per subroutine
  - ❍ A useful way to find out memory leaks

- □ The latest feature is to watch arrays (or contiguous pieces of memory) being accidentally overwritten
  - ❍ Finds the routine which does the overwrite
  - ❍ Checking is done by checking against 4-byte CRC32 cryptographic key for each watch-region

Motivation of having Dr.Hook

- Upon error IFS sometimes hangs and doesn't print any information about where the failure occurred

- May print misleading traceback from a non-computational thread, like (typical to nearly every Unix-system):

```
0:  Signal received: SIGINT - Interrupt


0:  Traceback:
0:    Location 0x0000377c
0:    Offset 0x00000868 in procedure pm_async_thread
0:    Offset 0x000000a4 in procedure _pthread_body
0:    --- End of call chain ---
```

Dr.Hook traceback

- When Dr.Hook is enabled, the traceback is much more informative, indented and up to date than system trbk
  - depends on program's Dr.Hook instrumentation level

```
0:[myproc#1,tid#1,pid#90320]: Received signal#2 (SIGINT) ; Memory: 219145K …
0:[myproc#1,tid#1,pid#90320]:  MASTER
0:[myproc#1,tid#1,pid#90320]:   CNT0
0:[myproc#1,tid#1,pid#90320]:    SU0YOMB
0:[myproc#1,tid#1,pid#90320]:     SUPHY
0:[myproc#1,tid#1,pid#90320]:      SUPHEC
0:[myproc#1,tid#1,pid#90320]:       SUECRAD
0:[myproc#1,tid#1,pid#90320]:        RRTM_KGB7
```

Dr. Hook - an instrumentation tool

How to instrument a Fortran90 program ?

```
    SUBROUTINE SUB
    USE YOMHOOK, ONLY : LHOOK, DR_HOOK
    IMPLICIT NONE

    REAL(8) :: ZHOOK_HANDLE ! Must be a local (stack) variable

!- The very first statement in the subroutine
    IF (LHOOK) CALL DR_HOOK(‘SUB’,0,ZHOOK_HANDLE)

!--- Body of the routine goes here ---

!- Just before RETURNing from the subroutine
    IF (LHOOK) CALL DR_HOOK(‘SUB’,1,ZHOOK_HANDLE)

    END SUBROUTINE SUB
```

How to instrument a C-program ?

```
#include “drhook.h”  /* ifsaux/include/drhook.h” */
/* You normally still need a Fortran90 main program ☹ */
void subname( )
{
  {
    DRHOOK_START(subname);

    /* or
    DRHOOK_START_BY_STRING(“subname”);
    */

    /* Body of the routine goes here */

    DRHOOK_END(0);
  }
}
```

Dr.Hook profiling information

- □ When Dr.Hook is enabled, it can also be asked to gather wall-clock (or CPU-time) information about routines being instrumented
- □ Profile is printed at exit, one (text)file per MPI-task :

```
Profiling information for program='./MASTER' (# of routines=506):
        Wall-time is 2.75 sec on proc#1 (2 procs, 3 threads)
 % time   cumul    self    total  # of calls    self    total   routine@<tid> [cluster:(id,size)]
 (self)   (sec)    (sec)   (sec)               ms/call  ms/call
  15.59   0.43     0.43    0.43          7      61.17    61.17   OPDIS@1 [134,1]
  12.11   0.76     0.33    0.33         64       5.20     5.20   POSNAM@1 [139,1]
   3.21   0.85     0.09    0.09          3      29.42    29.42   PPOPEN@1 [148,1]
   3.09   0.93     0.08    0.10      10916       0.01     0.01  *CUADJTQ@3 [28,3]
   3.07   0.93     0.08    0.09      10479       0.01     0.01   CUADJTQ@1 [28,3]
   3.04   0.93     0.08    0.09      10474       0.01     0.01   CUADJTQ@2 [28,3]
   3.00   1.02     0.08    0.12          2      41.17    62.15   WROUTSPGB@1 [498,1]
   2.80   1.09     0.08    0.08          1      76.82    81.32   SUSPECG@1 [421,1]
```

Dr.Hook profiling information (cont'd)

- □ When Mflop/s counter is enabled, the following output can be produced:

```
Profiling information for program='/fdb/eg7t/bin/ifsMASTER', myproc#1 (# of instrumented routines called = 859):
        Instrumentation started : 20031201 171315
        Instrumentation   ended : 20031201 173631
        Wall-time is 1247.54 sec on proc#1, 401 MFlops (ops#500104*10^6), 1358 MIPS (ops#1694634*10^6) (32 procs, 4 threads)
        Thread#1:     1241.66 sec (99.53%), 124 MFlops (ops#153788*10^6), 605 MIPS (ops#751376*10^6)
        Thread#2:      505.01 sec (40.48%), 228 MFlops (ops#115265*10^6), 622 MIPS (ops#314268*10^6)
        Thread#3:      504.12 sec (40.41%), 229 MFlops (ops#115330*10^6), 626 MIPS (ops#315331*10^6)
        Thread#4:      502.39 sec (40.27%), 230 MFlops (ops#115722*10^6), 624 MIPS (ops#313659*10^6)

    #  % Time        Cumul        Self       Total     # of calls    MIPS  MFlops   Div-%    Routine@<tid> [Cluster:(id,size)]
       (self)        (sec)        (sec)       (sec)

    1   10.23      127.564     127.564     170.783          8930     685      49    0.0   *CTXGETDB@1 [57,4]
    2    5.35      194.311      66.747      98.825       7257296     807     251    0.2   *VEXP_@2 [843,4]
    3    5.35      194.311      66.688      99.131       7290992     819     255    0.2    VEXP_@4 [843,4]
    4    5.34      194.311      66.614      98.761       7298576     812     252    0.2    VEXP_@1 [843,4]
    5    5.33      194.311      66.477      98.596       7295024     808     251    0.2    VEXP_@3 [843,4]
    6    4.81      254.324      60.013     116.628       2773222     643     307    5.6   *CUADJTQ@2 [60,4]
    7    4.80      254.324      59.925     116.691       2793808     639     305    5.6    CUADJTQ@1 [60,4]
```

Status of Dr.Hook with IFS (now CY29R2)

- □ Dr.Hook resides in library libifsaux.a
  - o In standalone Dr.Hook and/or ODB installations in libdrhook.a
- □ The CY28 was the first IFS-cycle, where the almost the whole suite had been instrumented with Dr.Hook
  - o Instrumentation can be done automatically with Perl-script
- □ In CY28R1 Dr.Hook had improved performance and due to this low basic overhead, the calling tree-tracer was switched ON by default on our operational environment
- □ In CY28R2 had much cheaper Mflop/s-rate monitoring in CY28R2+ we had much more calls instrumented
- □ CY28R4 saw memory profiling & CY29R2 watch points

Dr.Hook environment variables

- Enable Dr.Hook (call-tree/traceback only → cheap)
  - DR_HOOK=1

- Enable wall-clock time profiling information upon exit
  - DR_HOOK_OPT=prof
  - The profile will be written to files drhook.prof.<1..nproc>

- Redirect the profile-file to /path/file.<1..nproc>
  - DR_HOOK_PROFILE=/path/file

- Restrict output to MPL-task MYPROC=1
  - DR_HOOK_PROFILE_PROC=1

- Collect HPM (Mflop/s & MIPS) information
  - DR_HOOK_OPT=hpmprof or mflops

Dr.Hook environment variables (cont'd)

- □ Collect CPU-profile information
  - ○ DR_HOOK_OPT=cpuprof
- □ Print profiling information from routines that consume (self) at least (say) 0.5% of the total time
  - ○ DR_HOOK_PROFILE_LIMIT=0.5
- □ Collect memory and CPU-time information
  - ○ DR_HOOK_OPT="memory,cputime"
- □ Collect wall-clock time, heap & stack
  - ○ DR_HOOK_OPT="wall heap stack"
- □ Create memory profile & wall clock profile separately
  - ○ DR_HOOK_OPT="wallprof,memprof"

Dr.Hook environment variables (cont'd)

- □ Catch also Unix-signal number 1 (=SIGHUP)
  - ○ DR_HOOK_CATCH_SIGNALS=1

- □ Ignore Unix-signal 8 (=SIGFPE) from Dr.Hook
  - ○ DR_HOOK_IGNORE_SIGNALS=8

- □ Instead of including just the instrumented subroutine name as an entry in the profile, all calling trees of that routine (up to certain depth; def.=50) can be included as distinct callpath entries in profile:
  - ○ DR_HOOK_OPT="wallprof,callpath"
  - ○ DR_HOOK_CALLPATH_DEPTH=5
  - ○ Use sparingly → currently lots of overhead

How to get an instantaneous calling tree ?

```
INTEGER(4) :: IOUNIT, ITID, IOPT, INDENT
 INTEGER(4),EXTERNAL :: GET_THREAD_ID

IOUNIT    = 0    !   Fortran I/O-unit , say stderr
 ITID       = GET_THREAD_ID( )  ! 1 .. numthreads
 IOPT       = 2
 INDENT   = 0   !   Modified during the call

CALL C_DRHOOK_PRINT(IOUNIT, ITID, IOPT, INDENT)

! After this the variable INDENT equals to no. of routines
   seen in the traceback
```

Activating Dr.Hook system signal handler only

- You should enforce catching of Unix signals, even if DR_HOOK has not been set to 1

- It is highly recommended to have the following call

  ```
  CALL C_DRHOOK_INIT_SIGNALS(1)
  ```

  after MPI-initialization

- Although this may not provide you Dr.Hook's own call-trace upon abnormal exit (i.e. you had DR_HOOK=0), it would still try to produce the system specific traceback – this is often better than nothing

An example of Dr.Hook watch point

```
USE yomhook, ONLY : LHOOK, DR_HOOK
USE yomwatch
IMPLICIT NONE
REAL(8) :: ZHOOK
INTEGER B(1), ARRAY(100)
COMMON /AREA/ B,ARRAY
ARRAY(1:100) = 1
CALL DR_HOOK_WATCH ('ARRAY',ARRAY,LDABORT=.TRUE.)
CALL DR_HOOK('WATCH_SECTION',0,ZHOOK)
B(1:10) = 0 ! Bang!! Overwrites the 9 first elements of ARRAY, too
! Next Dr.Hook call inline detects the overwrite and aborts
CALL DR_HOOK('WATCH_SECTION',1,ZHOOK)
```

Dr.Hook availability

- □ With Mflop/s (HPM-)monitor
  - ○ IBM Power4 (by John Hague/Bob Walkup)
  - ○ Cray X1 (by Bob Carruthers)

- □ Other platforms (without HPM) i.e. tried on these :
  - ○ IBM Power3
  - ○ Linux (Pentium & AMD Opteron)
  - ○ SGI/MIPS
  - ○ Fujitsu VPP5000

- □ Portable to virtually any Unix-platform

Conclusion

- Dr.Hook has become an invaluable tool for ECMWF to
  - Detect programming errors
  - Find out performance statistics and especially Mflop/s
  - Chase memory leaks and memory overwrites

- ECMWF operational & research IFS forecasting and 4DVAR environments have DR_HOOK set to 1 all the time despite minor overheads, since
  - Upon failure we at least normally get a very accurate traceback, and a hunch on what might have gone wrong

- Dr.Hook will also help us in computer benchmarking, since we can now reliably compare performance profiles information between different vendors