Dr.Hook instrumentation tool

Sami Saarinen, Mats Hamrud, Deborah Salmond & John Hague

ECMWF, Shinfield Park, Reading
RG2 9AX, United Kingdom
sami.saarinen@ecmwf.int

ABSTRACT

Dr.Hook is a simple, low-overhead instrumentation tool, which allows you to keep track of dynamic calling tree of a program and print it in the event of failure. It can also gather performance profiling information on a per subroutine basis, which can be useful in estimating computational costs.

1 Introduction

The name Dr.Hook stems from Fujitsu VPP’s hook-function – a compiler option which allowed routines to be intercepted upon entry and exit by a user defined "hook"-function. The Dr.Hook-environment, unlike Fujitsu’s system, is meant to be portable across all Unix-systems, controlled by environment variables without need to recompile or relink.

In the recent years it has become more difficult to get reliable traceback upon failure. One reason is our complex environment and use of multiple MPI-tasks and OpenMP-threads. Error messages from such processes are often lost or can be quite misleading. As result much extra time has to be put into debugging. We wanted to reduce this laborious debugging and instead (or in addition to) offer a more direct way of trapping errors.

Since the CY28 of IFS was a cleaning cycle, it was decided to insert automatically calls to \texttt{DR\_HOOK()} - function while entering and leaving any IFS/ARPEGE related subroutine, a few thousand in total. This way the Dr.Hook-environment could keep track of dynamic calling tree and upon failure produce informative Dr.Hook-traceback.

As result of these calls we can hopefully fix programming errors much quicker than before without need to add extra print statements or wonder for several days what may have gone wrong.

A nice ”by-product” of Dr.Hook-instrumentation is that we can also collect profiling information on resources we spend in each instrumented routine. That is to say, we can collect routine call-counts, wall & CPU-times, chase for memory peaks, check paging activities and so on. Furthermore we can produce a user-friendly report at the end of execution per each MPI-task spread across every computational OpenMP-thread, if necessary.

A recent addition (CY28R1) to Dr.Hook system is possibility to obtain MFlop/s-rates per OpenMP-thread and MPI-task on the IBM Power4 machine. This greatly helps us to understand computational costs in IFS/ARPEGE and may even aid benchmarking.

Dr.Hook is currently callable from both Fortran90 and C-routines.
2 Instrumentation

Every instrumented routine (here: just a generic SUBNAME) has the following coding norm:

- at the top of the routine: enable access to YOMHOOK-module

  USE YOMHOOK, ONLY : LHOOK, DR_HOOK

- at the beginning of the routine (before the first executable statement)

  REAL(KIND=JPRB) :: ZHOOK_HANDLE
  IF (LHOOK) CALL DR_HOOK('SUBNAME', 0, ZHOOK_HANDLE)

- just before END- or before any RETURN- or CONTAINS-statements you must call

  IF (LHOOK) CALL DR_HOOK('SUBNAME', 1, ZHOOK_HANDLE)

The first call to DR_HOOK() adjusts the current dynamic calling tree by adding a new member to it. The last call to DR_HOOK() removes routine ‘SUBNAME’ from this instantaneous calling tree.

If in addition performance profiling is on, then the Dr.Hook-system collects exact (as opposed to sampled) performance profile on behalf of routine ‘SUBNAME’ and its descendants while in this routine.

If SUBNAME is a Fortran90 module procedure, then the name will contain the MODULE-name followed by a colon before the routine name (for example: ‘GFL_SUBS:DEFINE_GFL_COMP’).

It is important to note that the variable ZHOOK_HANDLE must be a local (stack) variable – i.e. not a SAVED variable. Its size has to be 8-bytes and the subroutine prototype requires it to be a REAL(8) floating point number. Upon return from the first DR_HOOK()-call it contains the address information to the Dr.Hook internal data structure, where accounting information for this particular routine (and of this OpenMP-thread) is kept. The last call to DR_HOOK() in the routine will use the value of variable ZHOOK_HANDLE as an address to locate and update routine’s accounting records.

If ZHOOK_HANDLE gets overwritten between these two calls to DR_HOOK(), then the second call will pick this it up immediately and raise an abort. This is an indication of either a genuine (stack variable) overwrite, or more often that some descendant routine has RETURNed before calling second time its DR_HOOK(). The latter case is easy to fix: just search for alternative exits and RETURN-statements from the descendant routines and check whether any path to the second call of DR_HOOK() exists at all.

The logical variable LHOOK is by default set to .TRUE. in module YOMHOOK. When the very first call to DR_HOOK() takes place in a program, then all Dr.Hook-environment variables are examined. If it turns out that Dr.Hook-facility was not turned on, then the logical variable LHOOK is set to .FALSE. and any further dynamic calling tree processing will be disabled.

In the future there will be available a checker-program to ensure that DR_HOOK() calls are entered correctly.
3 Environment variables

Environment variables are used to activate and drive Dr.Hook features. By default – if no variables are set – Dr.Hook facility is turned off. All values (except filenames) are case insensitive in the following table.

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
<th>Values &amp; remarks</th>
</tr>
</thead>
<tbody>
<tr>
<td>DR_HOOK</td>
<td>Enable/disable Dr.Hook</td>
<td>true or 1 activates Dr.Hook false or 0 is the default</td>
</tr>
<tr>
<td>DR_HOOK_OPT</td>
<td>A comma-separated list of:</td>
<td>Traceback will contain call counts</td>
</tr>
<tr>
<td></td>
<td>calls or count</td>
<td>Traceback contains CPU-times</td>
</tr>
<tr>
<td></td>
<td>cputime or cpu</td>
<td>Traceback contains wall clock times</td>
</tr>
<tr>
<td></td>
<td>walltime or wall</td>
<td>Union of cputime and walltime</td>
</tr>
<tr>
<td></td>
<td>times or time</td>
<td>Heap memory high water mark per routine</td>
</tr>
<tr>
<td></td>
<td>heap or hwm</td>
<td>Stack size monitoring per routine</td>
</tr>
<tr>
<td></td>
<td>stack or stk</td>
<td>Paging activity monitoring per routine</td>
</tr>
<tr>
<td></td>
<td>rss</td>
<td>Union of heap, stack, rss &amp; paging</td>
</tr>
<tr>
<td></td>
<td>paging or pag</td>
<td>Union of memory and times</td>
</tr>
<tr>
<td></td>
<td>memory or mem</td>
<td>Activates wall clock time based profiling</td>
</tr>
<tr>
<td></td>
<td>all</td>
<td>Implies also calls and walltime</td>
</tr>
<tr>
<td></td>
<td>prof or wallprof</td>
<td>Deactivates cpuprof</td>
</tr>
<tr>
<td></td>
<td>cpuprof</td>
<td>Activates CPU-time based profiling</td>
</tr>
<tr>
<td></td>
<td>hpmprof or hpm or mflops</td>
<td>Deactivates wallprof</td>
</tr>
<tr>
<td></td>
<td>trim</td>
<td>Activates MFlop/s profiling</td>
</tr>
<tr>
<td></td>
<td>self</td>
<td>Allow case insensitive input of routine name</td>
</tr>
<tr>
<td></td>
<td>noself</td>
<td>Very expensive! Don’t use this!</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Include Dr.Hook in the profile output, too</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Exclude Dr.Hook altogether from the profile</td>
</tr>
<tr>
<td>DR_HOOK_PROFILE</td>
<td>Filename(s) for the profile</td>
<td>One profile-file per MPI-task</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Use full path with .%d for MPL-task:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>/path/file.%d,%d = 1.%NPES</td>
</tr>
<tr>
<td></td>
<td></td>
<td>The default: drhook.prof.%d</td>
</tr>
<tr>
<td></td>
<td></td>
<td>If the suffix (.%d) is missing, it will be added</td>
</tr>
<tr>
<td></td>
<td></td>
<td>automatically</td>
</tr>
<tr>
<td></td>
<td></td>
<td>The default = -1 i.e. all MPL-tasks</td>
</tr>
<tr>
<td></td>
<td></td>
<td>The default = -10.0 (!)</td>
</tr>
<tr>
<td>DR_HOOK_PROFILE_PROC</td>
<td>MPL-task that produces profile</td>
<td></td>
</tr>
<tr>
<td>DR_HOOK_PROFILE_LIMIT</td>
<td>Routines that consume at least</td>
<td></td>
</tr>
<tr>
<td></td>
<td>this many percentages of (self-)time</td>
<td>resources will appear in the profile</td>
</tr>
<tr>
<td></td>
<td>resources will appear in the profile</td>
<td></td>
</tr>
</tbody>
</table>

Currently any of the options DR_HOOK_OPT="stack,paging,mflops" are only available for IBM Power4 machines, but ports to Cray SV2 (for example) are underway.
In addition the following environment variables are available to Dr.Hook, too:

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
<th>Values &amp; remarks</th>
</tr>
</thead>
<tbody>
<tr>
<td>DR_HOOK_CATCH_SIGNALS</td>
<td>A comma separated list of additional signals to be caught</td>
<td>The default = 0 i.e. no effect&lt;br&gt;Value = -1 activates all possible signals&lt;br&gt;The default = 0 i.e. no effect&lt;br&gt;Value = -1 deactivates all possible signals</td>
</tr>
<tr>
<td>DR_HOOK_IGNORE_SIGNALS</td>
<td>A comma separated list of signals not to be caught</td>
<td></td>
</tr>
<tr>
<td>DR_HOOK_HASHBITS</td>
<td>No. bits for hashing algorithm</td>
<td>The default = 15 i.e. allocates a hash data structure of $2^{15}$ elements. Between 1..24. Increase only if you have thousands of routines to monitor</td>
</tr>
</tbody>
</table>

4 Caught signals

The following table shows what Unix-signals are caught by Dr.Hook-system. Please note that signal SIGXPU (i.e. CPU-time exceeded) does not produce any profiling output, since it would cause Dr.Hook to catch SIGXPU infinitely, since there is no CPU-time left! Otherwise profile is written even if signal has been caught, since we believe it can still contain invaluable information.

<table>
<thead>
<tr>
<th>Signal name</th>
<th>Description</th>
<th>Remarks</th>
</tr>
</thead>
<tbody>
<tr>
<td>SIGABRT</td>
<td>Abort execution</td>
<td></td>
</tr>
<tr>
<td>SIGBUS</td>
<td>Bus error</td>
<td></td>
</tr>
<tr>
<td>SIGSEGV</td>
<td>Segmentation violation</td>
<td></td>
</tr>
<tr>
<td>SIGILL</td>
<td>Illegal instruction</td>
<td></td>
</tr>
<tr>
<td>SIGEMT</td>
<td>EMT instruction</td>
<td>N/A on Linux</td>
</tr>
<tr>
<td>SIGSTKFLT</td>
<td>Stack fault</td>
<td>Linux only; from 28R2 onwards</td>
</tr>
<tr>
<td>SIGFPE</td>
<td>Floating-point exception</td>
<td>Should be switched off when debugging</td>
</tr>
<tr>
<td>SIGTRAP</td>
<td>Trace trap</td>
<td></td>
</tr>
<tr>
<td>SIGINT</td>
<td>Interrupt</td>
<td></td>
</tr>
<tr>
<td>SIGQUIT</td>
<td>Quit</td>
<td></td>
</tr>
<tr>
<td>SIGTERM</td>
<td>Termination</td>
<td></td>
</tr>
<tr>
<td>SIGIO</td>
<td>I/O now possible</td>
<td>Typo; must be SIGIOT (fixed in CY28R2)</td>
</tr>
<tr>
<td>SIGXCPU</td>
<td>CPU limit exceeded</td>
<td>Ignores atexit() i.e. no profiling</td>
</tr>
<tr>
<td>SIGSYS</td>
<td>Bad system call</td>
<td></td>
</tr>
</tbody>
</table>

For actual signal numbers, please refer to your man signal-command or have look at your Unix-system’s include file /usr/include/signal.h (often in /usr/include/sys/signal.h). These are needed if you want to use environment variables DR_HOOK_CATCH_SIGNALS or DR_HOOK_IGNORE_SIGNALS to activate/deactivate some signals.

Very often catching a SIGSEGV means serious memory overwrite, a SIGBUS that some subroutine argument is missing or stack is corrupted, a SIGILL that illegal instruction is executed – say – corrupted or null function call is attempted, a SIGFPE floating point arithmetic with uninitialized or invalid numbers, a SIGINT that control-C is pressed (for interactive jobs), and so on.
5 Overhead

When dynamic calling tree is activated (but nothing else) overhead of DR_HOOK() -calls seem to be around 1% in the current IFS/ARPEGE configurations on ECMWF’s IBM Power4. This seems to be a low price to pay for enabling correct tracebacks, and thus reducing need for extensive debugging and head scratching.

When the wall clock time profiling is on, overheads seem to be around 5%. And if the MFlop/s-rate counters are activated, the overhead is still bearable 10-15%. And all these extra profiling options can be turned on via environment variables, without need to recompile or relink anything.

6 Some auxiliary routines

You can activate Dr.Hook’s signal catching feature to be able to produce at least system specific traceback more reliably by calling routine C_DRHOOK_INIT_SIGNALS():

CALL C_DRHOOK_INIT_SIGNALS(1)

This instructs Dr.Hook’s signal handling functions to be on the top of the list of the functions to be called in the event of failure. And if Dr.Hook-facility was turned off you will not get its dynamic calling tree, but system specific traceback, which may or may not be accurate. See for example ifs/setup/sumpini.F90.

When Dr.Hook has been activated, you can print instantaneous calling tree at any moment to a Fortran I/O-unit by calling routine C_DRHOOK_PRINT():

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 .. # of OpenMP-threads
IOPT = 2 ! Print current calling tree
INDENT = 0 ! Indentation; modified during the call

CALL C_DRHOOK_PRINT(IOUNIT, ITID, IOPT, INDENT)

! The variable INDENT now equals to no. of routines seen in the traceback
7 C-interface

Also C-routines can be instrumented under Dr.Hook-system. The interface (in CY28R1) to C is as follows:

```c
#include "drhook.h" /* from "ifsaux/include/drhook.h" */

void subname()
{
    /* Variable declarations */

    DRHOOK_START(subname); /* as the first call to DR_HOOK() in Fortran90 */

    /* or as a constant (compile-time evaluable) character string:
       DRHOOK_START_BY_STRING("subname"); */

    /* The first actual executable statement */

    /* The body of the routine "subname" goes here */

    /* The last thing ;
       the zero (0) below can be replaced with some size-information,
       like the number of bytes processed or so */

    DRHOOK_END(0);
    return;
}
```

You can see first examples in odb/lib/codb.c and odb/include/codb.h. Please note that symbols
DRHOOK_START, DRHOOK_START_BY_STRING and DRHOOK_END are all simple and cheap C-macros,
which take care of declaring the Fortran90 ZHOOK_HANDLE-equivalent inside the macros.

8 Source code and libraries

In CY28R1, Dr.Hook resides in IFSAUX, files ifsaux/support/drhook.c and include file
ifsaux/include/drhook.h and is therefore naturally built into library libifsaux.a. The Fortran90-
part resides in files ifsaux/module/yomhook.F90 and ifsaux/support/dr_hook_*.F90.

Timers and memory routines needed are found mostly under ifsaux/utilities/-directory. Some other
resource related routines are still found (for historical reasons) under odb/aux/util_ccode.c, but can
be found from file ifsaux/utilities/util_timers.c in CY28R2.

Dummies for HPM (High Performance Monitoring) for IBM are found under odb/lib/Dummies.c, in case
you hit problems with unsatisfied externals.

On IBM Power4 machine in order to activate the HPM-feature, you need to compile the file
ifsaux/support/drhook.c with option -DHPM and link executable either with dummies
(-lodbdummy i.e. no runtime HPM) or with HPM-libraries (at ECMWF the $LIBHPM equals to
-L/usr/pmapi/lib -lpmapi).
9 Future enhancements

After CY28R1 we will investigate ways to incorporate profiling of data movement. That is to say, we want to get clearer picture on amounts of data transferred across MPI-processes and to be able to trace I/O-bottlenecks. Furthermore we want to be able to show the filename of the routine being called. This is important for especially C-routines, where there maybe several routines sitting the same file making searching process more difficult.

The Fortran90-interface to \texttt{DR\_HOOK}() will in the future contain two more (optional) arguments: size information (an \texttt{INTEGER(4)}) and the (source) filename parameters. In fact, the C-interface in cycle CY28R1 already contains this functionality and some ODB C-routines already benefit from these two extra arguments. As an example, you could pass the number of bytes retrieved from a database, or decide that you want to see how many rows of data satisfy a particular SQL-query condition. In both cases you could also opt for source filename of the routine in concern.

The basic Dr.Hook without fancy MFlop/s-monitors, but with traceback feature, will run at least on the following machines in CY28R2: IBM Power4 & Power3, Pentium/Linux and Silicon Graphics. We still have to check how Fujitsu VPP5000 version will behave. Also port to Cray SV2 is underway; for example traceback can already be produced correctly.

10 Known bugs or features

Dr.Hook-system cannot handle recursive function calls correctly i.e. profiling information for such routines (and calling tree before the routine) will be incorrect. This will be fixed in the future. Meantime, remove Dr.Hook from potentially recursive routines.

When a failure occurs on a non-master OpenMP-thread (i.e. $>1$), the dynamic calling tree incorrectly prints the full current calling tree of the master-thread before proceeding to print thread’s own tree. Although this doesn’t matter too much, you will still see slightly misleading traceback and duplicate output of some routines that were active in both master and slave thread during failure. This will fixed in the future, too.

The signal SIGIOT has mistakenly typed as SIGIO. The SIGIOT normally equals to SIGABRT. If you believe that this incorrect coding causes problems and you want to remove SIGIO (normally #29) from the list of Dr.Hook-catchable signals, you can ignore this signal via environment variable as follows (Korn-shell):

\begin{verbatim}
export DR_HOOK_IGNORE_SIGNALS=29
\end{verbatim}

This typo will be fixed in CY28R2.

Acknowledgements

Thanks to Bob Walkup from IBM Watson Research for information about how to read the HPM-counters on IBM Power4 machine.
Examples of Dr.Hook output from IFS:

Dr. Hook Traceback

0: 19:57:40 0:00 STEF 7:0 corresponds to: CPU time limit exceeded
1: Traceback
2: -location: 000000000000000000
3: 0:000000000000000000 in procedure _eventstep
4: 0:000000000000000000 in procedure _signal
5: 0:000000000000000000 in procedure _nssync_thread
6: --- End of call chain ---

Dr. Hook for T511 forecast

| # | Time | Curnul Self Total | # of calls | MIPS | KILOPS | Div* | Footnote<1>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>(self)</td>
<td>(sec)</td>
<td>(sec)</td>
<td>(sec)</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
| 1 | 4:45 | 185.463 | 185.463 | 185.463 | 127.729 | 737 | 48 | VKN (32, 6)
| 2 | 14:09 | 126.852 | 126.852 | 126.852 | 126.852 | 728 | 48 | VKN (32, 6)
| 3 | 14:09 | 126.852 | 126.852 | 126.852 | 126.852 | 728 | 48 | VKN (32, 6)
| 4 | 5:16 | 164.207 | 164.207 | 164.207 | 164.207 | 753 | 47 | VKN (32, 6)
| 5 | 5:09 | 120.504 | 120.504 | 120.504 | 120.504 | 735 | 47 | VKN (32, 6)
| 6 | 5:16 | 120.504 | 120.504 | 120.504 | 120.504 | 735 | 47 | VKN (32, 6)
| 7 | 4:45 | 120.504 | 120.504 | 120.504 | 120.504 | 735 | 47 | VKN (32, 6)
| 8 | 9:09 | 120.504 | 120.504 | 120.504 | 120.504 | 735 | 47 | VKN (32, 6)
| 9 | 9:09 | 120.504 | 120.504 | 120.504 | 120.504 | 735 | 47 | VKN (32, 6)
| 10 | 9:09 | 120.504 | 120.504 | 120.504 | 120.504 | 735 | 47 | VKN (32, 6)
| 11 | 9:09 | 120.504 | 120.504 | 120.504 | 120.504 | 735 | 47 | VKN (32, 6)
| 12 | 9:09 | 120.504 | 120.504 | 120.504 | 120.504 | 735 | 47 | VKN (32, 6)
| 13 | 9:09 | 120.504 | 120.504 | 120.504 | 120.504 | 735 | 47 | VKN (32, 6)
| 14 | 9:09 | 120.504 | 120.504 | 120.504 | 120.504 | 735 | 47 | VKN (32, 6)
| 15 | 9:09 | 120.504 | 120.504 | 120.504 | 120.504 | 735 | 47 | VKN (32, 6)
| 16 | 9:09 | 120.504 | 120.504 | 120.504 | 120.504 | 735 | 47 | VKN (32, 6)
| 17 | 9:09 | 120.504 | 120.504 | 120.504 | 120.504 | 735 | 47 | VKN (32, 6)
| 18 | 9:09 | 120.504 | 120.504 | 120.504 | 120.504 | 735 | 47 | VKN (32, 6)
| 19 | 9:09 | 120.504 | 120.504 | 120.504 | 120.504 | 735 | 47 | VKN (32, 6)
| 20 | 9:09 | 120.504 | 120.504 | 120.504 | 120.504 | 735 | 47 | VKN (32, 6)
| 21 | 9:09 | 120.504 | 120.504 | 120.504 | 120.504 | 735 | 47 | VKN (32, 6)
| 22 | 9:09 | 120.504 | 120.504 | 120.504 | 120.504 | 735 | 47 | VKN (32, 6)
| 23 | 9:09 | 120.504 | 120.504 | 120.504 | 120.504 | 735 | 47 | VKN (32, 6)
| 24 | 9:09 | 120.504 | 120.504 | 120.504 | 120.504 | 735 | 47 | VKN (32, 6)
| 25 | 9:09 | 120.504 | 120.504 | 120.504 | 120.504 | 735 | 47 | VKN (32, 6)
| 26 | 9:09 | 120.504 | 120.504 | 120.504 | 120.504 | 735 | 47 | VKN (32, 6)
| 27 | 9:09 | 120.504 | 120.504 | 120.504 | 120.504 | 735 | 47 | VKN (32, 6)