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 laborous 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 `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 computional 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.

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

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:

Variable	Description	Values & remarks
DR_HOOK_CATCH_SIGNALS	A comma separated list of *additional* signals to be caught	The default = 0 i.e. no effect Value = -1 activates all possible signals
DR_HOOK_IGNORE_SIGNALS	A comma separated list of signals *not* to be caught	The default = 0 i.e. no effect Value = -1 deactivates all possible signals
DR_HOOK_HASHBITS	No. bits for hashing algorithm	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

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.

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

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 `SIGFPU` 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:

```
#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 `DR_HOOK()` will in the future contain two more (optional) arguments: size information (an `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 incorrecty 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 belive 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):

```
export DR_HOOK_IGNORE_SIGNALS=29
```

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:   15:57:40 STEP   936 H= 234:00 +CPU= 41.379
 13:[myproc#14,tid#4,pid#55924]: Received signal#24 (SIGXCPU) ; Memory: 2019178K (heap), 0K (stack)
 13:[myproc#14,tid#1,pid#55924]:  MASTER ,#1,st=1,wall=0.000s/0.000s
 13:[myproc#14,tid#1,pid#55924]:   CNT0 ,#1,st=1,wall=0.000s/0.000s
 13:[myproc#14,tid#1,pid#55924]:    CNT1 ,#1,st=1,wall=0.000s/0.000s
 13:[myproc#14,tid#1,pid#55924]:     CNT2 ,#1,st=1,wall=0.000s/0.000s
 13:[myproc#14,tid#1,pid#55924]:      CNT3 ,#1,st=1,wall=0.000s/0.000s
 13:[myproc#14,tid#1,pid#55924]:       CNT4 ,#1,st=1,wall=0.000s/0.000s
 13:[myproc#14,tid#1,pid#55924]:        STEP0 ,#978,st=1,wall=10531.259s/0.000s
 13:[myproc#14,tid#1,pid#55924]:         SCAN2H ,#1018,st=1,wall=8913.967s/0.043s
 13:[myproc#14,tid#1,pid#55924]:          SCAN2MDM ,#1018,st=1,wall=8913.896s/32.036s
 13:[myproc#14,tid#1,pid#55924]:           GP_MODEL ,#938,st=1,wall=8845.641s/4.830s
 13:[myproc#14,tid#1,pid#55924]:            EC_PHYS ,#213893,st=1,wall=6144.597s/22.378s
 13:[myproc#14,tid#1,pid#55924]:             CALLPAR ,#213893,st=1,wall=5856.788s/88.130s
 13:[myproc#14,tid#1,pid#55924]:              SLTEND ,#213893,st=1,wall=662.390s/179.559s
 13:[myproc#14,tid#1,pid#55924]:               CUADJTQ ,#117188599,st=1,wall=1992.364s/1477.382s
 13:[myproc#14,tid#4,pid#55924]:                EC_PHYS ,#213356,st=1,wall=5145.442s/22.418s
 13:[myproc#14,tid#4,pid#55924]:                 CALLPAR ,#213356,st=1,wall=5860.376s/88.000s
 13:[myproc#14,tid#4,pid#55924]:                  CUCALLN ,#213810,st=1,wall=2731.710s/27.983s
 13:[myproc#14,tid#4,pid#55924]:                   CUMASTRN ,#213810,st=1,wall=2679.495s/36.678s
 13:[myproc#14,tid#4,pid#55924]:                    CUDDRAFN ,#213810,st=1,wall=66.548s/23.442s
 13:
 13:  Signal received: SIGXCPU - CPU time limit exceeded
 13:
 13:  Traceback:
 13:    Location 0x0000377c
 13:    Offset 0x0000009c in procedure _event_sleep
 13:    Offset 0x00000318 in procedure sigwait
 13:    Offset 0x000006c8 in procedure pm_async_thread
 13:    Offset 0x000000a4 in procedure _pthread_body
 13:    --- End of call chain ---
```

Dr. Hook for T511 forecast

#	% Time (self)	Cumul (sec)	Self (sec)	Total (sec)	# of calls	MIPS	MFlops	Div-%	Routine@<tid> [Cluster:(id,size)]
1	12.15	147.001	147.001	197.794	11278293	714	371	4.3	*CUADJTQ@1 [32,4]
2	12.11	147.001	146.494	196.885	11258918	718	373	4.3	CUADJTQ@3 [32,4]
3	12.09	147.001	146.247	197.027	11286073	719	374	4.3	CUADJTQ@4 [32,4]
4	12.08	147.001	146.062	196.742	11278641	717	373	4.3	CUADJTQ@2 [32,4]
5	5.56	214.298	67.296	80.726	104	733	41	0.0	TRANSINV_MDL@1 [516,1]
6	5.56	281.552	67.254	83.603	97	931	277	2.9	WUCOUPLE@1 [557,1]
7	5.46	347.613	66.062	73.709	98	851	19	0.1	TRANSDIR_MDL@1 [514,1]
8	3.71	392.507	44.894	151.873	45384	959	31	3.4	*CUASCN@1 [33,4]
9	3.69	392.507	44.662	152.351	45414	960	31	3.4	CUASCN@2 [33,4]
10	3.69	392.507	44.643	152.440	45344	958	31	3.4	CUASCN@3 [33,4]
11	3.68	392.507	44.493	151.696	45458	968	31	3.4	CUASCN@4 [33,4]
12	3.12	430.218	37.711	89.444	22729	1065	347	4.2	*CLOUDSC@4 [5,4]
13	3.11	430.218	37.605	89.363	22707	1068	348	4.2	CLOUDSC@2 [5,4]
14	3.11	430.218	37.572	89.120	22672	1065	347	4.2	CLOUDSC@3 [5,4]
15	3.09	430.218	37.317	89.525	22692	1071	348	4.2	CLOUDSC@1 [5,4]
16	2.50	460.513	30.295	66.332	22729	1220	151	6.7	*CUBASEN@4 [34,4]
17	2.49	460.513	30.149	66.253	22707	1229	152	6.7	CUBASEN@2 [34,4]
18	2.48	460.513	30.034	66.113	22692	1230	152	6.7	CUBASEN@1 [34,4]
19	2.48	460.513	29.989	66.173	22672	1233	152	6.7	CUBASEN@3 [34,4]
20	2.17	486.730	26.217	37.878	2588598	832	46	13.3	*CUENTR@2 [42,4]
21	2.17	486.730	26.207	37.691	2584608	828	46	13.3	CUENTR@3 [42,4]
22	2.16	486.730	26.149	37.597	2591106	833	46	13.3	CUENTR@4 [42,4]
23	2.15	486.730	26.006	37.426	2586888	833	46	13.3	CUENTR@1 [42,4]
24	2.07	511.791	25.060	36.427	2584608	776	29	2.8	*CUBASMCN@3 [35,4]
25	2.06	511.791	24.909	36.436	2588598	779	28	2.8	CUBASMCN@2 [35,4]
26	2.05	511.791	24.737	36.194	2591106	785	29	2.8	CUBASMCN@4 [35,4]
27	2.04	511.791	24.704	36.128	2586888	772	27	3.0	CUBASMCN@1 [35,4]