

DR HOOK Instrumentation tool

January 20, 2009

Contents

References	1
1 Dr.Hook instrumentation tool	3

List of Tables

1	Dr.Hook environment variables	5
2	More Dr.Hook environment variables	6

1 Dr.Hook instrumentation tool

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.

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 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.

Instrumentation

Every instrumented routine (here: just a generic SUBR) 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('SUBR', 0, ZHOOK_HANDLE)
```

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

```
IF (LHOOK) CALL DR_HOOK('SUBR', 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 'SUBR' 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 'SUBR' and its descendants while in this routine.

If the SUBR 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 `SAVE`d 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 the 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 `RETURN`d 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.

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 1: Dr.Hook environment variables

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 cputime or cpu walltime or wall times or time heap or hwm stack or stk rss paging or pag memory or mem all prof or wallprof cpuprof hpmprof or hpm or mflops trim self noself	Traceback contains call counts Traceback contains CPU-times Traceback contains wall clock times Union of cputime and walltime Heap memory high water mark per routine Stack size monitoring per routine Resident set size monitoring Paging activity monitoring per routine Union of heap, stack, rss & paging Union of memory and times Activates wall clock time based profiling Implies also calls and walltime Deactivates cpuprof Activates CPU-time based profiling Implies also calls and cputime Deactivates wallprof Activates MFlop/s profiling Implies wallprof Allow case insensitive input of routine name Very expensive! Don't use this! Include Dr.Hook in the profile output, too Exclude Dr.Hook altogether from the profile By default Dr.Hook overhead is always,

continued on next page ...

Table 1: Dr.Hook environment variables ... *continued*

Variable	Description	Values & remarks
	nosize or nosizeinfo noprop or nopropagate cluster or clusterid	accounted for, but not printed in the profile output Do not display size-information Do not propagate signals outside Dr.Hook Display cluster-id information
DR_HOOK_HPMSTOP	Stops HPM-counting for the given routine after <i>ncalls</i> if <i>mflops</i> -rate has not been achieved	Set this to value <i>ncalls</i> or <i>ncalls,mflops</i>

Currently the options `DR_HOOK_OPT="stack,paging,mflops"` are only available for IBM Power4 machines, but ports to Cray X1 (for example) are underway. As of writing this document, Cray X1 hardware performance monitor for obtaining MFlop/s-rates has just been implemented.

In addition the following environment variables are also available to Dr.Hook:

Table 2: More Dr.Hook environment variables

Variable	Description	Values & remarks
DR_HOOK_PROFILE	Filename(s) for the profile	One profile-file per MPI-task Use full path with <code>.%d</code> for MPL-task: <code>/path/file.%d</code> , <code>%d = 1..\$NPES</code> The default: <code>drhook.prof.%d</code> If the suffix (<code>.%d</code>) 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 (!)
DR_HOOK_CATCH_SIGNALS	A comma separated list of <i>additional</i> 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 <i>not</i> to be caught	The default = 0 i.e. no effect. Value = -1 deactivates all possible signals

continued on next page ...

Table 2: More Dr.Hook environment variables ... *continued*

Variable	Description	Values & remarks
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

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 the profile is written even if a signal has been caught, since we believe it still can contain valuable 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	
SIGXCPU	CPU limit exceeded	Ignores <code>atexit()</code> 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 means that illegal instruction is executed, a SIGFPU means a floating point arithmetic on uninitialized or invalid numbers, a SIGINT means that control-C is pressed (for interactive jobs), and so on.

Overhead

When the dynamic calling tree is activated (but nothing else) the 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-lice scratching!

When the wall clock time profiling is on, overheads seem to be around 2%. 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. Please note that the HPM-overhead can now be effectively minimized by setting environment value `DR_HOOK_HPMSTOP` to (say) `20,500000`. This means: stop calling HPM-counters for any given routine after 20 calls unless routines performance exceeds 500000 Mflop/s (500GigaFlop/s!).

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 a 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 an 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
```

C-interface

Also C-routines can be instrumented under Dr.Hook-system. The interface (in CY28R2) to C is as follows:


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

void subname()
{
    /* Variable declarations */

    { /* Begin block (optional) */
        /* The following ia like the first call to DR_HOOK() in Fortran90 */

        DRHOOK_START(subname);

        /* 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);
    } /* End block (optional, if the Begin block also omitted) */
    return;
}
```

You can see examples in files odb/lib/codb.c and odb/include/codb.h. 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.

Source code and libraries

In CY28R2, Dr.Hook resides in IFS AUX, 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 the ifsaux/utilities/-directory. Some other resource related routines are still found (for historical reasons) under odb/aux/util_ccode.c, but can also be found in 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 an 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 called `$LIBHPM`) `-L/usr/pmapi/lib -lpmapi`.

Enhancements

After CY28R2 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. This has been partially done via `GSTAT`-subroutine (in `IFSAUX`).

Furthermore we want to be able to show the filename of the routine being called. This is important especially for C-routines, where there maybe several routines sitting the same file making searching process more difficult.

The Fortran90-interface to `DR_HOOK()` contains two more 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. Port to Cray X1 is nearly completed and it is now already possible to easily compare this machine's performance with IBM Power4 rather realistically.

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 mislead-

ing traceback and duplicate output of some routines that were active in both master and slave thread during failure.

The signal SIGIOT was 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):

```
$ export DR_HOOK_IGNORE_SIGNALS=29
```

This bug has now been 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.

Thanks to Bob Carruthers from Cray UK for supplying HPM-counters for Cray X1 machines.

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), OK (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=562.390s/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=6145.442s/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]: CUDDRAIN ,#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 0x00000068 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@ctid> [Cluster:(id,size)]
1	12.15	147.001	147.001	197.794	11278293	714	371	4.3	*CUADJTQ01 [32,4]
2	12.11	147.001	146.494	196.885	11258918	718	373	4.3	CUADJTQ03 [32,4]
3	12.09	147.001	146.247	197.027	11286073	719	374	4.3	CUADJTQ04 [32,4]
4	12.08	147.001	146.062	196.742	11278641	717	373	4.3	CUADJTQ02 [32,4]
5	5.56	214.298	67.296	80.726	104	733	41	0.0	TRANSINU_MDL01 [516,1]
6	5.56	281.552	67.254	83.603	97	931	277	2.9	WCCOUPLE01 [557,1]
7	5.46	347.613	66.062	73.709	98	851	19	0.1	TRANSDIR_MDL01 [514,1]
8	3.71	392.507	44.894	151.873	45384	959	31	3.4	*CUASCN01 [33,4]
9	3.69	392.507	44.662	152.351	45414	960	31	3.4	CUASCN02 [33,4]
10	3.69	392.507	44.643	152.440	45344	958	31	3.4	CUASCN03 [33,4]
11	3.68	392.507	44.493	151.696	45458	968	31	3.4	CUASCN04 [33,4]
12	3.12	430.218	37.711	89.444	22729	1065	347	4.2	*CLOUDSC04 [5,4]
13	3.11	430.218	37.605	89.363	22707	1068	348	4.2	CLOUDSC02 [5,4]
14	3.11	430.218	37.572	89.120	22672	1065	347	4.2	CLOUDSC03 [5,4]
15	3.09	430.218	37.317	89.525	22692	1071	348	4.2	CLOUDSC01 [5,4]
16	2.50	460.513	30.295	66.332	22729	1220	151	6.7	*CUBASEN04 [34,4]
17	2.49	460.513	30.149	66.253	22707	1229	152	6.7	CUBASEN02 [34,4]
18	2.48	460.513	30.034	66.113	22692	1230	152	6.7	CUBASEN01 [34,4]
19	2.48	460.513	29.989	66.173	22672	1233	152	6.7	CUBASEN03 [34,4]
20	2.17	486.730	26.217	37.878	2588598	832	46	13.3	*CUENTR02 [42,4]
21	2.17	486.730	26.207	37.691	2584608	828	46	13.3	CUENTR03 [42,4]
22	2.16	486.730	26.149	37.597	2591106	833	46	13.3	CUENTR04 [42,4]
23	2.15	486.730	26.006	37.426	2586888	833	46	13.3	CUENTR01 [42,4]
24	2.07	511.791	25.060	36.427	2584608	776	29	2.8	*CUBASMCN03 [35,4]
25	2.06	511.791	24.909	36.436	2588598	779	28	2.8	CUBASMCN02 [35,4]
26	2.05	511.791	24.737	36.194	2591106	785	29	2.8	CUBASMCN04 [35,4]
27	2.04	511.791	24.704	36.128	2586888	772	27	3.0	CUBASMCN01 [35,4]