SONY

On ARM

Measuring Function Duration with Ftrace

By Tim Bird
Sony Corporation of America
<tim.bird (at) am.sony.com>

SONY

Outline

- Introduction to Ftrace
- Adding function graph tracing to ARM
- Duration Filtering
  - Optimizing the discard operation
- Post-trace analysis tools
- Performance impact
- Resources

SONY

Introduction to Ftrace

- What is Ftrace?
- Overview of operation
  - Instrumentation
  - Runtime operation
  - Data capture
  - Trace log output
- Function graph tracing

SONY

What is Ftrace?

- Ftrace is the first generic tracing system to get mainlined (Hurray!!)
  - Mainlined in 2.6.27
  - Derived from RT-preempt latency tracer
- Provides a generic framework for tracing
  - Infrastructure for defining tracepoints
  - Ability to register different kinds of tracers
  - Specialized data structure (ring buffer) for trace data storage

SONY

Overview of FTrace Operation

- Instrumentation
  - Explicit
    - Tracepoints defined by declaration
    - Calls to trace handler written in source code
  - Implicit
    - Automatically inserted by compiler
      - Uses gcc ‘-pg’ option
    - Inserts call to ‘mcount’ in each function prologue
    - Easy to maintain – no source code modifications
    - Only practical way to maintain 20,000+ tracepoints

SONY

mcount Routine

- ‘mcount’ is called by every kernel function
  - Except inlines and a few special functions
- Must be a low-overhead routine
- Incompatible with some compiler optimizations
  - E.g. cannot omit frame-pointers on ARM
  - Compiler disables some optimizations automatically
  - Works with ARM EABI
  - Assembly analysis indicates that mcount callers have well-defined frames
- Misc note:
  - New mcount routine (_gnu_mcount) is coming

Code to Call mcount

```
00000570 <sys_sync>:
570: e1a0c00d mov ip, sp
574: e92dd800 stmdb sp!, {fp, ip, lr, pc}
578: e24cb004 sub fp, ip, #4 ; 0x4



57c: e3a00001 mov r0, #1 ; 0x1
580: ebffffa0 bl 408 <do_sync>
584: e3a00000 mov r0, #0 ; 0x0
588: e89da800 ldmia sp, {fp, sp, pc}
```

```
00000570 <sys_sync>:
570: e1a0c00d mov ip, sp
574: e92dd800 stmdb sp!, {fp, ip, lr, pc}
578: e24cb004 sub fp, ip, #4 ; 0x4
57c: e1a0c00e mov ip, lr
580: ebffffe bl 0 <mcount>
584: 00000028 andeq r0, r0, r8, lsr #32
588: e3a00001 mov r0, #1 ; 0x1
58c: ebffff9d bl 408 <do_sync>
590: e3a00000 mov r0, #0 ; 0x0
594: e89da800 ldmia sp, {fp, sp, pc}
```

SONY

Trace setup at run-time

- Pseudo-files in debugfs
  - e.g. mount debugfs –t debugfs /debug
- Select a tracer
  - e.g. echo function_graph >current_tracer
- Set tracing parameters
  - e.g. echo 100 >tracing_threshhold
  - echo funcgraph-abstime >trace_options

SONY

Trace Data Capture

- Ring Buffer
  - Specialized structure for collecting trace data
    - Manages buffer as list of pages
  - Latest version is lockless for writing
    - Ability to atomically reserve space for an event
  - Automatic timestamp management
  - Per-cpu buffers
    - Avoids requiring cross-CPU synchronization
    - Also avoids cache collisions
      - Very important for performance

SONY

Trace Output

- Output is human readable text
  - No special tools required to collect trace data
- Examples:
  - cat trace
    - Returns EOF at end of trace data
  - cat trace_pipe | grep foo >log.txt
    - Blocks at end of trace data
- Quick enable/disable
  - echo 0 >tracing_enabled

SONY

Ring Buffer Operations

- ring_buffer_lock_reserve
  - Atomically reserve space in buffer
- ring_buffer_event_data
  - Get pointer to place to fill with data
- ring_buffer_unlock_commit
  - Commit event data
- ring_buffer_discard_commit
  - Discard reserved data space

SONY

Function graph tracing

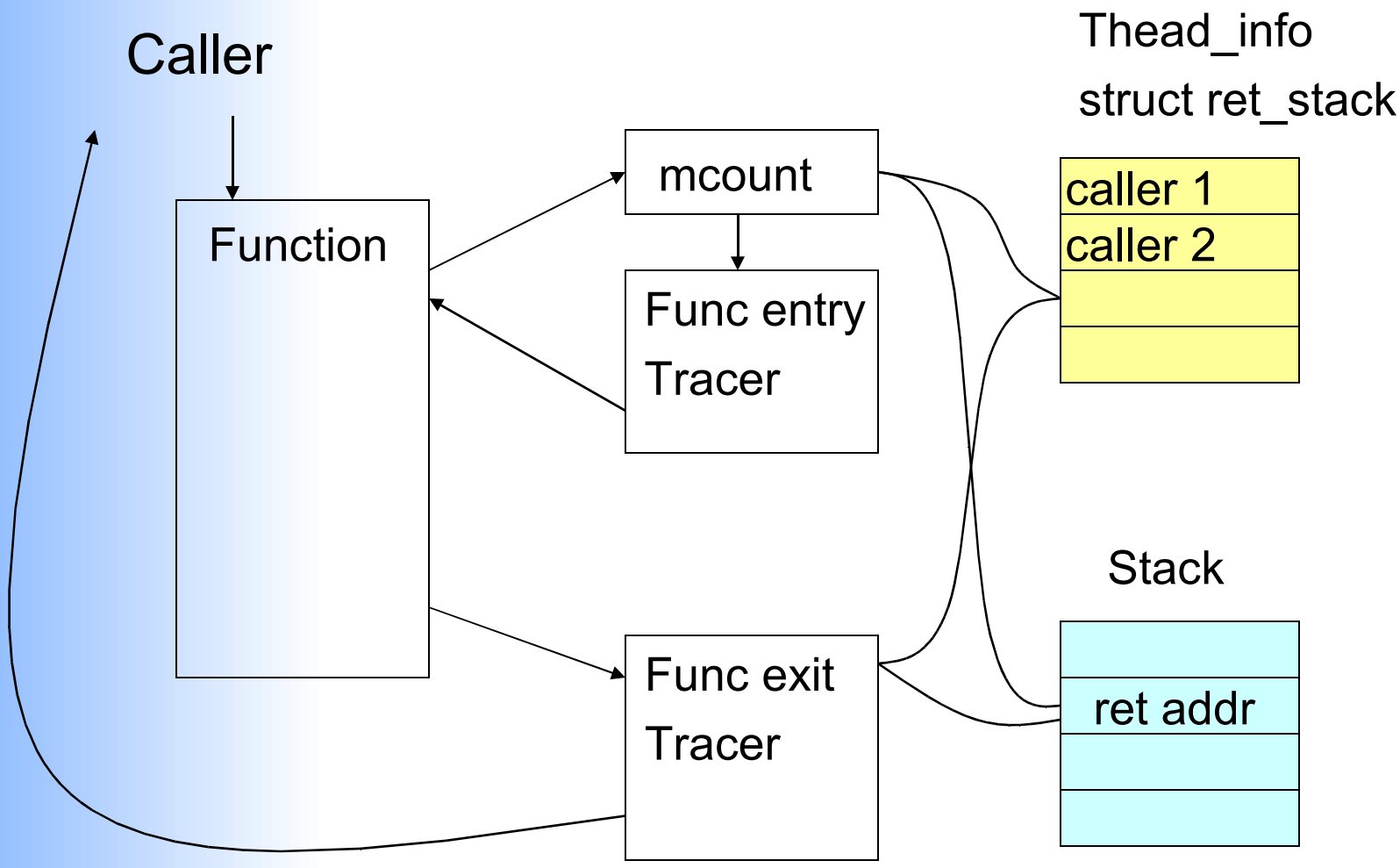
- Traces function entry and exit
- What is it good for?
  - See relationship between functions
    - Is a GREAT way to learn about kernel
    - Find unexpected/abnormal code paths
  - Measure function duration
    - Find long latencies and performance problems
- But, the -pg option only instruments function entry

SONY

Hooking function exit

- Normal ‘function’ tracer just traces function entry capture
- To capture function exit, a trampoline is used
  - mcount:
    - Saves real return address
    - Replaces return address with address of trampoline
  - In exit tracer, return to the real return address

Diagram of Trampoline



SONY

Filtering by Duration

- Compare duration to threshhold
- Discard function entry and exit events
- Easy to discard exit event
  - Just don’t commit data
- Trickier to discard entry event
  - ring_buffer_event_discard() converts event to padding if subsequent events have been committed to buffer
    - Wastes a lot of space
    - Severely constrains the time coverage for a trace

SONY

Optimizing Event Discard

- Normally, can't discard events after other events are committed to buffer
- However, with duration filtering, if an event is filtered for duration, then all children functions are filtered also
- "Last event" in buffer is always function entry for current exit
  - Only have to "rewind" one event, which is relatively easy (and likely safe)

SONY

Results from optimized discard

Discard operation	Duration Filter Value	Total Function Count	Time covered by Trace	Trace event count
Discard_event	0	3.292M	0.39 s	27392
Discard_event	1000	3.310M	1.29 s	26630
Discard_event	100000	3.309M	1.34 s	26438
Rewind_tail	0	3.295M	0.39 s	27316
Rewind_tail	1000	3.327M	31.26 s	35565
Rewind_tail	100000	3.328M	79.44s †	1669

† The test only lasted 79 seconds—extrapolating the results yields a trace coverage time of 27 minutes

SONY

Example of Use

```
$ mount debugfs -t debugfs /debug
$ cd /debug/tracing
$ cat available_tracers
function_graph function sched_switch nop
$ echo 0 >tracing_enabled
$ echo 1000 >tracing_thresh
$ echo function_graph >current_tracer
$ echo 1 >tracing_enabled
$ for i in `seq 1 10` ; do ls /bin | sed s/a/z/g ; done
$ echo 0 >tracing_enabled
$ echo funcgraph-abstime >trace_options
$ echo funcgraph-proc >trace_options
$ cat trace
```

Function Graph Results

```
# tracer: function_graph
#
#      TIME        CPU  TASK/PID         DURATION                  FUNCTION CALLS
#       |          |     |    |           |   |                     |   |   |   |
  193.719625 |   0)     ls-556     |               |  sys_lstat64() {
  193.719641 |   0)     ls-556     |               |    vfs_lstat() {
  193.719650 |   0)     ls-556     |               |      vfs_fstatat() {
  193.719660 |   0)     ls-556     |               |        user_path_at() {
  193.719722 |   0)     ls-556     |               |          do_path_lookup() {
  193.719755 |   0)     ls-556     |               |            path_walk() {
  193.719777 |   0)     ls-556     |               |              __link_path_walk() {
  193.719826 |   0)     ls-556     |               |                do_lookup() {
  193.719855 |   0)     ls-556     |               |                  nfs_lookup_revalidate() {
  193.719883 |   0)     ls-556     | ! 1028.500 us |                    _text();
  193.719946 |   0)     ls-556     | ! 1189.500 us |                  }
  193.719965 |   0)     ls-556     | ! 1258.500 us |                }
  193.719986 |   0)     ls-556     | ! 1775.167 us |              }
  193.720016 |   0)     ls-556     | ! 1874.333 us |            }
  193.720045 |   0)     ls-556     | ! 2018.167 us |          }
  193.720069 |   0)     ls-556     | ! 2143.000 us |        }
  193.720099 |   0)     ls-556     | ! 2397.000 us |      }
  193.720108 |   0)     ls-556     | ! 2415.167 us |    }
  193.720139 |   0)     ls-556     | ! 2478.334 us |  }
  193.720315 |   0)     ls-556     |               |  sys_lstat64() {
  193.720337 |   0)     ls-556     |               |    vfs_lstat() {
  193.720346 |   0)     ls-556     |               |      vfs_fstatat() {
  193.720357 |   0)     ls-556     | ! 1094.500 us |        user_path_at();
  193.720410 |   0)     ls-556     | ! 1738.167 us |      }
  193.720419 |   0)     ls-556     | ! 1758.500 us |    }
  193.720452 |   0)     ls-556     | ! 1825.500 us |  }
```

SONY

Post-trace analysis

- Using ftd to analyze data
  - Measuring function counts
  - Measuring “local time”
    - wall time minus sub-routine wall time
    - May be wrong if we block
      - Need an option to subtract time that function was scheduled out
  - Filter, sort, select output columns,etc.

Ftd Output

Function	Count	Time	Average	Local
schedule	59	1497735270	25385343	1476642939
sys_write	56	1373722663	24530761	2892665
vfs_write	56	1367969833	24428032	3473173
tty_write	54	1342476332	24860672	1212301170
do_path_lookup	95	1076524931	11331841	34682198
__link_path_walk	99	1051351737	10619714	6702507
rpc_call_sync	87	1033211085	11875989	1700178
path_walk	94	1019263902	10843233	3425163
rpc_run_task	87	960080412	11035407	2292360
rpc_execute	87	936049887	10759194	2316635
__rpc_execute	87	932779083	10721598	11383353
do_lookup	191	875826405	4585478	9510659
call_transmit	100	785408085	7854080	5871339
__nfs_revalidate_inode	38	696216223	18321479	1652173
nfs_proc_getattr	38	690552053	18172422	1234634

Performance issues

- Overhead of tracing
  - Can be substantial
    - Average function duration = 1.72 µs
    - Overhead = 18.89 microseconds per function
  - Test used was CPU-bound
    - find /sys >/dev/null
    - With I/O bound test, ratio of overhead to average function length should be much lower

SONY

Overhead Measurements

Tracer Status	Elapsed Time	Function count	Time per function	Overhead per function
TRACE=n	9.25 s	2.91M	1.72 us	-
Nop	10.30 s	2.92M	2.05 us	0.33 us
Graph disabled	19.85 s	2.98M	5.22 us	3.50 us
Graph active	72.15 s	3.29M	20.61 us	18.89 us

SONY

Roadmap and future work

- Mainline stuff
  - ARM function graph tracing
  - Duration filtering
    - Recently rejected – back to the drawing board??
- Need to use functionality to improve bootup time

SONY

Measuring kernel boot

- Requirements for using ftrace in early boot
  - Availability of clock source
  - Static(?) definition of trace parameters
    - Start location for tracing (optimally start_kernel)
  - Initialization of ring buffer and tracer registration
    - Would be nice to do at compilation time, but that's hard!

SONY

References

- Ftrace tutorial at OLS 2008
  - http://people.redhat.com/srostedt/ftrace-tutorial.odp
  - Video: http://free-electrons.com/pub/video/2008/ols/ols2008-steven-rostedt-ftrace.ogg
- “The world of Ftrace” at Spring 2009 LF Collaboration Summit
  - http://people.redhat.com/srostedt/ftrace-world.odp
- Patches and tools for this talk
  - http://elinux.org/Ftrace_Function_Graph_ARM

SONY

Q & A