Hardware-assisted software tracing

Adrien Vergé
adrienverge@gmail.com
talk about tracing
improve tracing using hardware
1 Tracing
2 Hardware
3 Improvements
1 Tracing
“a technique used to understand what is going on in a system in order to debug or monitor it”
recording events

from the kernel:  IRQ handlers, system calls, scheduling activity, network activity, etc.

in user-space:  tracepoints inside your application
Why is my software crashing?
Where are the bottlenecks?
How to improve performance?

use less resources
run faster
save battery
a process spawns 2 threads:
#1 produces chunks of data that #2 consumes
### Process Timeline

<table>
<thead>
<tr>
<th>Process</th>
<th>TID</th>
<th>PTID</th>
<th>Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>bottleneck</td>
<td>26242</td>
<td>26226</td>
<td>12:40:48.500</td>
</tr>
<tr>
<td>bottleneck</td>
<td>26243</td>
<td>26242</td>
<td>12:40:48.600</td>
</tr>
<tr>
<td>bottleneck</td>
<td>26244</td>
<td>26242</td>
<td>12:40:48.700</td>
</tr>
</tbody>
</table>

### CPU and IRQ Activity

- **CPU 0**
- **CPU 1**
- **CPU 2**
- **CPU 3**
- **CPU 4**
- **CPU 5**
- **CPU 6**
- **CPU 7**
- **IRQ 44**
- **IRQ 46**
- **IRQ 43**
- **SOFT_IRQ 9**
- **SOFT_IRQ 4**
- **SOFT_IRQ 1**
- **SOFT_IRQ 7**

Colors:
- **WAIT_BLOCKED**
- **WAIT_FOR_CPU**
- **USERMODE**
- **SYSCALL**
- **INTERRUPTED**
### Table

<table>
<thead>
<tr>
<th>Process</th>
<th>TID</th>
<th>PTID</th>
</tr>
</thead>
<tbody>
<tr>
<td>bottleneck</td>
<td>26242</td>
<td>26226</td>
</tr>
<tr>
<td>bottleneck</td>
<td>26243</td>
<td>26242</td>
</tr>
<tr>
<td>bottleneck</td>
<td>26244</td>
<td>26242</td>
</tr>
</tbody>
</table>

### Diagram

- **WAIT_BLOCKED**
- **WAIT_FOR_CPU**
- **USERMODE**
- **SYSCALL**
- **INTERRUPTED**

- Execve event at 12:40:48.500
- E event at 12:40:48.600
- C event at 12:40:48.700

---

**Example:** LTTng+TMF
<table>
<thead>
<tr>
<th>Process</th>
<th>TID</th>
<th>PTID</th>
</tr>
</thead>
<tbody>
<tr>
<td>bottleneck</td>
<td>26242</td>
<td>26226</td>
</tr>
<tr>
<td>bottleneck</td>
<td>26243</td>
<td>26242</td>
</tr>
<tr>
<td>bottleneck</td>
<td>26244</td>
<td>26242</td>
</tr>
</tbody>
</table>

**Diagrams:**
- **WAIT_BLOCKED**
- **WAIT_FOR_CPU**
- **USERMODE**
- **SYSCALL**
- **INTERRUPTED**
tracing:

recording events for use in further analysis

[16.246595] Bluetooth: HCI device and connection manager up
[16.246602] Bluetooth: HCI socket layer initialized
[16.246605] Bluetooth: L2CAP socket layer initialized
[16.246609] Bluetooth: SCO socket layer initialized
[16.317299] Bluetooth: RFCOMM socket layer initialized
[16.317303] Bluetooth: RFCOMM sockets layer initialized
[16.317306] Bluetooth: RFCOMM version 1.2
[16.496886] Bluetooth: BNEP (Ethernet Emulation) ver 1.3
[16.496897] Bluetooth: BNEP socket layer initialized
[17.045998] NFSD: Using /var/lib/nfs/v4recovery as the NFSv4 state recovery directory
[17.046487] NFSD: starting 90-second grace period (net ffffffff81886100)
[17.327576] e1000e 0000:00:19.0: irq 42 for MSI/MSI-X
[17.430960] e1000e 0000:00:19.0: irq 42 for MSI/MSI-X
[17.431056] IPv6: ADDRCONF(NETDEV_UP): eth0: link is not ready
[19.452905] e1000e: eth0 NIC Link is Up 100 Mbps Half Duplex, Flow Control: Rx/Tx
[19.452910] e1000e 0000:00:19.0 eth0: 10/100 speed: disabling TSO
[21.344148] systemd-logind[3544]: Watching system buttons on /dev/input/event2 (Power Button)
[21.344198] systemd-logind[3544]: Watching system buttons on /dev/input/event3 (Video Bus)
[21.344242] systemd-logind[3544]: Watching system buttons on /dev/input/event1 (Power Button)
[31.982187] systemd-logind[3544]: Linked /tmp/.X11-unix/X0 to /run/user/1000/X11-display.
[37.509891] FAT-fs (sdb1): utf8 is not a recommended IO charset for FAT filesystems, filesystem will
tracing: recording events for use in further analysis

So it's just logging?
tracing vs. logging

compact binary trace format
buffering — avoid disk IO
lockless algorithms
low-level optimizations

result: \(~200\ \mu s\ vs. \sim 200\ \text{ns} / \text{event}\)
tracing users

real-time

OPAL-RT

Autodesk

CAE

Montavista

Adobe

Google

IBM

Freescale

STMicroelectronics

Siemens

Wind River

Embedded Systems

Nokia

Ericsson

CAE

OPAL-RT

Autodesk

Montavista

Adobe

Google

IBM

Freescale

STMicroelectronics

Siemens

Wind River

Embedded Systems

Nokia

Ericsson

heavy workload

intrusion detection

real-time
Beyond Heisenberg: observe without altering

- perform **light** (size) and **fast** (time)
- don't pollute **memory** space
- **thousands** of events / s
2 Hardware
Microchips are no longer just CPUs
Lots of tracing units

Freescale (PowerPC)
Nexus Program Trace,
Data Acquisition...

ARM
CoreSight
ETM, ETB, STM...

Intel (x86)
BTS, LBR, PT...
lots of tracing units

STM (event tracing)

ETM (execution tracing)

BTS (execution tracing)
lots of tracing units

supported by (probably good)
proprietary software
widely spread

Do you have one of these?
widely spread

Is your Intel CPU newer than this one?
3

Improvements
3 Improvements

1/3 STM on ARM
System Trace Module (STM)

Goal: help software recording events
System Trace Module (STM)

Provides dedicated resources:
- bus
- buffer
- timestamping

Need to instrument software
System Trace Module (STM)

system-on-chip

CPU

ETM

STM

ETB

system bus

timestamping
“LTTng-equivalent”

The traced process is instrumented: calling `tracepoint()` writes to the STM.

Embedding payload is possible.

A consumer process retrieves generated traces and stores them.
Traces are encoded in STP. Optimized, compact but proprietary format.
indicative benchmark: overhead mostly depends on the traced application!

- **no tracing**
- **LTTng-UST**
- **STM + ETB**

Bar chart showing:
- **time per iteration (µs)**

- **only tracepoints**
- **computation + tracepoints**
results

- no tracing
- LTTng-UST
- STM + ETB

<table>
<thead>
<tr>
<th>Time per iteration (µs)</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
</tr>
<tr>
<td>5</td>
</tr>
<tr>
<td>10</td>
</tr>
<tr>
<td>15</td>
</tr>
<tr>
<td>20</td>
</tr>
</tbody>
</table>

- only tracepoints
- computation + tracepoints

- `clock_gettime`
- `sched_getcpu`
3

Improvements

2/3

ETM on ARM
Embedded Trace Macrocell (ETM)

Goal: trace execution
Embedded Trace Macrocell (ETM)

Goal: trace execution
i.e. save every executed instruction address
Embedded Trace Macrocell (ETM)

Provides dedicated resources

address comparators, buffer, timestamping
Embedded Trace Macrocell (ETM)

Can focus on a specific process or function triggers upon custom conditions

Provides dedicated resources address comparators, buffer, timestamping
Embedded Trace Macrocell (ETM)

Can focus on a specific process or function triggers upon custom conditions

Provides dedicated resources
address comparators, buffer, timestamping

No need to instrument software
Embedded Trace Macrocell (ETM)
ETM not meant to trace events
Idea:

**do execution tracing on event addresses**

set address comparators to trigger in [event, event+4]

ETM not meant to trace events
implementation

needed to write
kernel support for
process and
function tracing
results

The graph shows the time per iteration (µs) for different tracing options.

- **no tracing**
- **LTTng-UST**
- **ETM + ETB**

There is an event loss indicated at the bottom of the graph, with a red arrow pointing to the bars representing more computation.
3

Improvements

3/3 BTS on x86
Branch Trace Store (BTS)

Goal: trace execution
Branch Trace Store (BTS)

x86 host

CPU

BTS

RAM

branch records

40ef26 4015a8
4015b0 7f2aac77e012
7f2aac77e024 4015b4
Branch Trace Store (BTS)
does not provide dedicated buffers
cannot focus on a specific process or function: traces every branch!
$ perf record -e branches:u -c 1 -d ./myprogram
$ perf script -f time,ip,addr
101918.272364: ffffffff814a6f2c => 7f8d7b9b3180
101918.272364: ffffffff814a6f2c => 7f8d7b9b3180
101918.272364: 7f8d7b9b3183 => 7f8d7b9b6730
101918.272364: ffffffff814a6f2c => 7f8d7b9b6730
101918.272364: ffffffff814a6f2c => 7f8d7b9b674f
101918.272364: ffffffff814a6f2c => 7f8d7b9b6756
101918.272364: ffffffff814a6f2c => 7f8d7b9b67df
101918.272364: 7f8d7b9b67c2 => 7f8d7b9b67c8
101918.272364: 7f8d7b9b67e3 => 7f8d7b9b67c8
101918.272364: 7f8d7b9b67e3 => 7f8d7b9b6a30
101918.272364: 7f8d7b9b67e3 => 7f8d7b9b6a58
101918.272364: 7f8d7b9b67e3 => 7f8d7b9b6bc0
101918.272364: 7f8d7b9b6bd7 => 7f8d7b9b67d3
101918.272364: 7f8d7b9b67e3 => 7f8d7b9b67c8
101918.272364: 7f8d7b9b67e3 => 7f8d7b9b67c8
101918.272364: 7f8d7b9b67e3 => 7f8d7b9b67c8
101918.272364: 7f8d7b9b67e3 => 7f8d7b9b67c8
101918.272364: 7f8d7b9b67e3 => 7f8d7b9b67c8
101918.272364: 7f8d7b9b67e3 => 7f8d7b9b67c8
101918.272364: 7f8d7b9b67e3 => 7f8d7b9b67c8
101918.272364: 7f8d7b9b67e3 => 7f8d7b9b67c8
BTS not meant to trace events if enabled, traces every branch

"Is hardware-assisted branch tracing faster than pure-software event tracing?"
hardware-traced with BTS:

simple program, every branch recorded

software-traced with LTTng:

same program, add a tracepoint() at every branch
results

![Graph showing the program branching rate (branch/s) and time per iteration (µs) for different tracing methods.]

- **no tracing**
- **LTTng-UST**
- **BTS with perf**

**Program branching rate (branch/s)**
- 2.1 x 10^8
- 2.4 x 10^7
- 1.1 x 10^7
- 7.6 x 10^6
- 5.5 x 10^6
- 4.4 x 10^6
- 3.7 x 10^6
- 3.2 x 10^6

**Time per iteration (µs)**
- 0.1
- 0.2
- 0.3
- 0.4
- 0.5
- 0.6
- 0.7
- 0.8
BTS writes trace to a dedicated buffer

trace is copied to a bigger memory zone upon buffer full or context switch

user stores trace to disk using the write system call

possible copy in another buffer because no O_SYNC flag

original perf
new “spliced” perf

BTS writes trace to a dedicated buffer

upon buffer full or context switch, move to the next sub-buffer

filled sub-buffers are labeled to be written to disk later

writing is done by a kernel task in user context

disk
results

Program branching rate (branch/s)

Time per iteration (µs)

- no tracing
- LTTng-UST
- BTS with perf
- BTS with "spliced" perf
Results

STM
-75% overhead compared to LTTng-UST
needs post-decoding

ETM
-30% to -50% overhead
limited number of tracepoints
no payload

BTS
not suited for event tracing (not flexible)
compared to vanilla perf, 2× faster
Freescale: Data Acquisition Program Trace

Intel: Processor Trace
last words
tracing helps you build efficient software
using LTTng:
very low footprint

Cortex-A9: ~5 µs / event
Core i7: ~200 ns / event
using hardware: almost zero footprint trace in production!
Links

LTTng and TMF:
https://lttng.org/

STM libraries:
https://github.com/adrienverge/libcoresightomap4430

ETM patch:
https://lkml.org/lkml/2014/1/30/259

BTS patch:
https://github.com/adrienverge/linux/tree/patch_perf_bts_splice
Thank you

Questions?