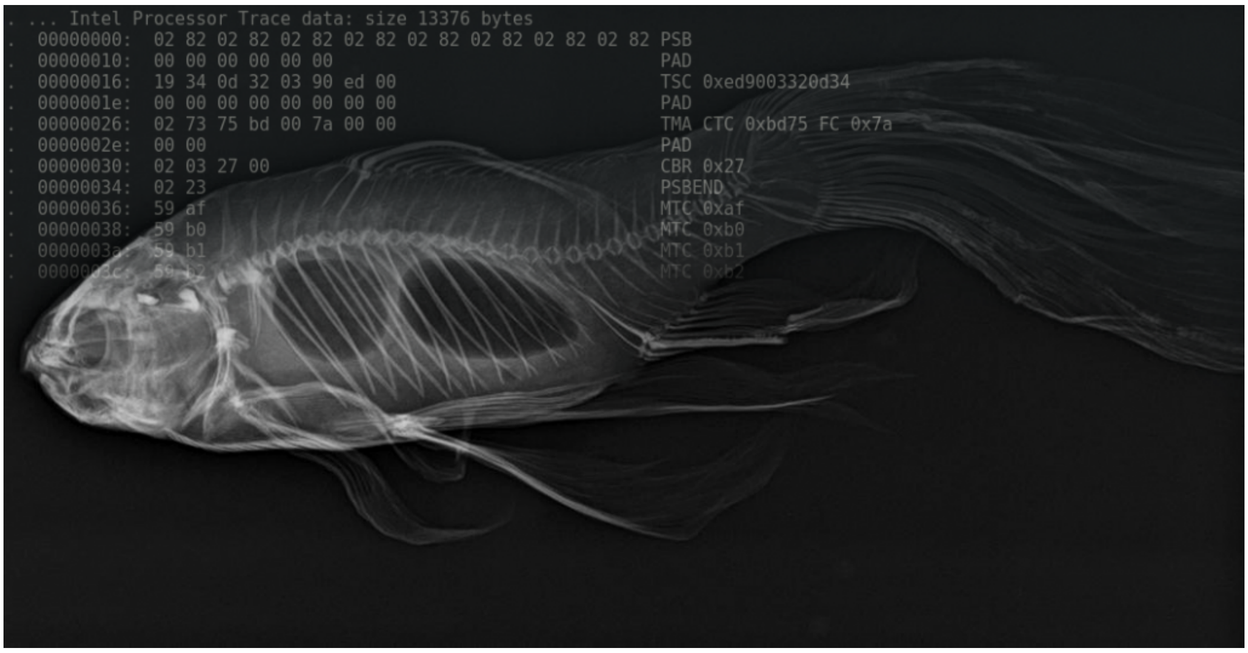
MONITORING

Hardware Tracing for Fast and Precise Performance Analysis

12 Oct 2015 2:14pm, by Suchakra Sharma



```
. ... Intel Processor Trace data: size 13376 bytes
.  00000000:  02 82 02 82 02 82 02 82 02 82 02 82 02 82 02 82 PSB
.  00000010:  00 00 00 00 00 00                               PAD
.  00000016:  19 34 0d 32 03 90 ed 00                         TSC 0xed9003320d34
.  0000001e:  00 00 00 00 00 00 00 00                         PAD
.  00000026:  02 73 75 bd 00 7a 00 00                         TMA CTC 0xbd75 FC 0x7a
.  0000002e:  00 00                                           PAD
.  00000030:  02 03 27 00                                     CBR 0x27
.  00000034:  02 23                                           PSBEND
.  00000036:  59 af                                           MTC 0xaf
.  00000038:  59 b0                                           MTC 0xb0
.  0000003a:  59 b1                                           MTC 0xb1
.  0000003c:  59 b2                                           MTC 0xb2
```

+

I have discussed the importance of tracing before — how it is an indispensable tool for in-depth analysis of userspace and kernel applications, especially when there is a need for an extremely low overhead, yet highly detailed, view of the system. Tools like LTTng, Ftrace and SystemTap have proved their mettle quite well with varying levels of satisfaction. For quite some time, processors and systems on chips (SoCs) have provided some dedicated hardware tracing blocks to give some edge to the developers when they struggle with complex bugs. Lets have a look at some of the hardware tracing infrastructure that has been used before and some recent interesting developments in this area.

Specialized hardware for debugging and tracing has been used in the past. This trend has been more prominent in the embedded systems world, however, rather than desktop or server-grade machines. For quite some time, chips have supported Joint Test Action Group (JTAG) standards with special ports that can be used to control debugging and tracing remotely. These are usually interfaced with in-circuit emulators that utilize in-built debugging support in the processor hardware.

Sponsor Note
October 14-16, 2015: The world's largest group of APM experts will assemble at Perform 2015 to discuss digital performance management, share best practices and network with thought leaders. The New Stack's Alex Williams will be on site. Be sure to catch up with him and get involved in the coverage and hear a key panel with the industry's best.

dynatrace
PERFORM
2015

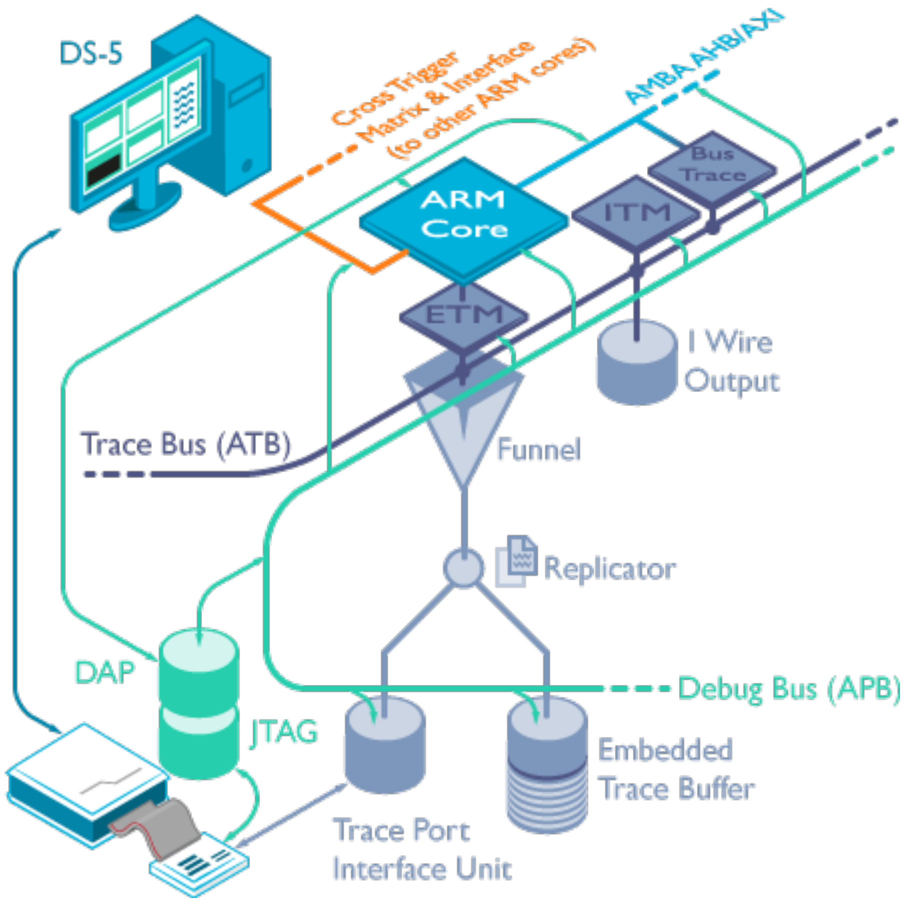
As an example, ARM has had support for Embedded In-Circuit Emulator (EmbeddedICE) hardware as early as the ARM7 and ARM9 variants. The EmbeddedICE logic was able to provide direct access to the data and address bus of the CPU. The debugging process can be controlled through the TAP controller, which was interfaced to the external world with the help of special debugging devices through the JTAG interface. These devices would then be used in conjunction with debugging software on the host machine.

Eventually, newer architectures, such as ARM11 and Cortex, provided support for Embedded Trace Macrocell for non-invasive, real-time tracing. This idea took hold and eventually evolved into what we now refer to as CoreSight which provides debug and trace support in the ARM world.

There are special products, such as TRACE32 Tools and TimeMachine Debugging Suite, which are built around this special hardware support in modern ARM processors. Lets have a look at hardware tracing support in two most important architectures: ARM and Intel x86.

ARM CoreSight

ARM's CoreSight debug and trace architecture is an almost complete solution in itself. Hardware components, like Embedded Trace Macrocell (ETM) and Program Trace Macrocell (PTM), are interfaced with an AMBA Advanced Trace Bus (ATB). Based on settings in configuration registers, trace data can be stored in the Embedded Trace Buffer (ETB) for consumption. The trace control is provided with the debug bus through standard JTAG interfaces.



Source : ARM CoreSight

Hardware tracing generates huge amounts of data — in the MB per second range. Through the debug bus access points, JTAG or CoreSight connectors — and the use of special hardware, like DSTREAM — the developers can access this huge stream of trace data. The DSTREAM unit is an external hardware device that interfaces with the ARM development platform and the trace support in ARM DS-5 Development Studio. Similar special hardware is also provided by both Lauterbach and Green Hills Software in the form of tools like SuperTrace Probe, which interfaces with their own respective development tools (which like ARM, are proprietary).

There is, however, a growing interest in open source tools. Considering the importance of Linux in the embedded world, Linaro, along with ARM, has been working on getting CoreSight support into the Linux kernel — it seems to be taking good shape and looks quite accessible. It is now possible to control CoreSight interfaces using sysfs, and then extracting data easily from the trace buffer (ETB). Though the analysis is meant for tools like DS-5 and TRACE32, if you feel adventurous you can use ptm2human to decode the program trace data (PTM provides program flow tracing, direct/indirect branches exceptions, etc.)

Intel Processor Trace

Similar to ARM's approach, Intel has pushed hard to get hardware tracing to a presentable state in recent years. For quite some time — since the Pentium 4 days — there has been a provision in Intel to record the program flow by using Last Branch Record (LBR), where breakpoints at each branch taken, interrupt or exception could be set. This was further extended to dedicated hardware in the chip which would record each and every taken branch information in a Branch Trace Store (BTS) buffer.

BTS, even though not so popular when introduced, eventually formed the basis for a more complete execution of flow tracing infrastructure in the form of Intel Processor Trace (Intel PT). A lot of gory detail on Intel PT has been mentioned in Chapter 36 of the Intel 64 and IA-32 Architectures Software Developer Manual. Basically, the main goal of Intel PT is to control flow tracing by tracking branches in code. Intel PT configuration can be done by using certain model-specific registers (MSRs). When the tracing is enabled, the processor starts generating and collecting trace information in the form of packets. Trace data in each packet corresponds to branching hardware "events." The current hardware trace packets can collect information, such as:

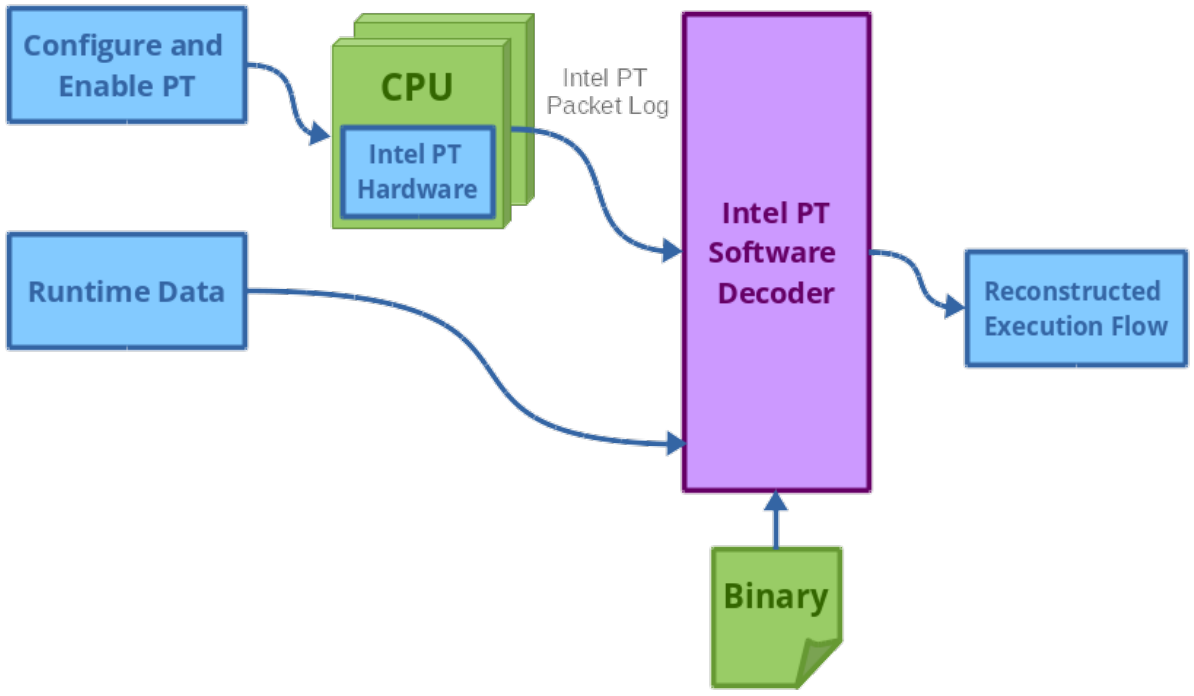
- Paging information (changed CR3 values)
- Time stamps
- Taken and not-taken information (tracking conditional branch directions)

The trace packets can also record the target Instruction Pointer (IP) of branches, exceptions and interrupts, and source IP for asynchronous events (such as exceptions and interrupts). There is a provision for filtering the trace based on IP range, specific logical processor, CR3 value range, etc. Tracing can be triggered based on the address range of the IP, branching status or filter status, or it can be triggered manually.

Trace output can be written in various formats, such as on direct contiguous physical address spaces, or variable sized regions stored as tables of pointers to those regions. There is also an option for a platform-specific trace transport scheme, depending on the support available on the platform. Apart from that, there's a "Cycle Accurate" mode, which adds a cycle-counter value to each packet, so that we can do actual metrics to compute actual instruction-per-cycle values. There is just an immense amount of hardware trace information that can be collected. The question is, what can we do with it?

Making Intel PT Accessible

Software developers and performance engineers would be happy to know that hardware tracing is now being integrated with popular performance analysis and debugging tools like perf and GDB. Intel PT's support has already landed in the Linux kernel. With the release of kernel 4.2, userspace support for perf-based Intel PT tracing has also arrived. How does this all work?



Intel PT Tracing Flow

The hardware configuration for tracing is done by the a special perf driver in the kernel. This perf driver dumps the data in to a separate ring buffer, which is eventually decoded by the Intel PT decoder — part of the perf userspace tools. The perf output can be decoded by a user-developed, stand-alone tool, as well. This is done, especially, because the trace data is a high volume stream, and it makes perfect sense to offload the decoding part to the userspace (the kernel can't keep up with the huge data flow). This decoded trace data is used along with the information from the target binary and other associated event run-time data to reconstruct the actual flow of the program — instruction by instruction. This is what differentiates PT from traditional instruction profiling approaches, which are sampling-based. All of this can be achieved with a very low overhead too.

Here is a sample run which I just did on my new toy — a Skylake Intel Core i5-6600K machine — running Linux kernel 4.3.0-rc3 :

```
$ perf list | grep intel_pt
intel_pt//                                     [Kernel PMU event]
```

We see that the `intel_pt` event is available. Lets look at the hardware tracing for userspace application `date`. We need the debug symbols for the userspace application as well:

```
$ perf record -e intel_pt//u date
Sun Oct 11 11:35:07 EDT 2015
[ perf record: Woken up 1 times to write data ]
[ perf record: Captured and wrote 0.027 MB perf.data ]
```

Now we can use `perf report` or other fancy perf scripts to display the data. Here is what the perf report shows :

```
$ perf report
...
# Samples: 1  of event 'instructions:u'
# Event count (approx.): 157207
#
# Overhead  Command  Shared Object  Symbol
# ........  .......  .............  ..........................
```

```
#
   100.00%  date     libc-2.21.so   [.] _nl_intern_locale_data
               |
               ---_nl_intern_locale_data
                  _nl_load_locale_from_archive
                  _nl_find_locale
                  setlocale
...
```

We see that a total of 157,207 instructions were recorded by the tracer. You can try out the `perf report -D` command to see how the Processor trace data has been recorded. It is even possible to generate flame graphs from the perf data. To learn more, I highly recommend this article on LWN by Andi Kleen, and the upcoming Intel PT documentation in the kernel. For the more curious, Andi Kleen also has a tool called simple-pt, for more raw control over hardware tracing.

Conclusion

Hardware tracing is an important tool. It may not be so evident, but having dedicated hardware to perform very precise tracing changes how we view tracing generally, from a pure software perspective. Even though it's having a successful streak in the embedded world with ARM and MIPS, Intel PT aims to truly bring this to the masses. It is going to fulfill use-cases like dumping hardware trace on crashes; selectively enabling-disabling hardware tracing on poor performing server nodes for analysis; a better reproduction of execution flow of programs, rather than sampling, so that transient errors in long-running programs can be identified with confidence. There are exciting times ahead. Let's see how debug-analysis tools can leverage this hardware power!

Intel is a sponsor of The New Stack.

Feature image: Derived from Angelo's Xray by John Smith, licensed under CC BY 2.0.

ANALYSIS OPEN SOURCE

+

RELATED STORIES

DEVELOPMENT / KUBERNETES / SECURITY

The New Stack Context: Two Views of Open Source Security

21 Feb 2020 5:00pm, by Libby Clark

CULTURE / TECHNOLOGY

Why Bruce Perens Is Proposing 'Coherent Open Source'

5 Feb 2020 8:54am, by David Cassel

View / Add Comments

Please stay on topic and be respectful of others. Review our Terms of Use.

SPONSORED FEED

Securely Connect and Scale Remote Workforces
MARCH 11, 2020

InfluxData's Business Continuity Plan for COVID-19
MARCH 11, 2020

The Benefits of Remote Work Beyond Avoiding the Coronavirus (COVID-19)
MARCH 11, 2020

Debugging Event Sources for AWS Lambda
MARCH 11, 2020

Extend visibility into workload and cluster health by leveraging native Kubernetes events
MARCH 11, 2020

Why Large Organizations Trust Kubernetes
MARCH 11, 2020

Azure NetApp Files: The Perfect Recipe
MARCH 11, 2020

Red Hat and the value of sharing
MARCH 11, 2020

No More Tiers: Reimagining the Structure of SecOps
MARCH 11, 2020

HashiCorp Consul Service on Azure - Private Beta Preview
MARCH 10, 2020

Make tracking agreements simple with our new Compliance Dashboard
MARCH 10, 2020

The TARS Foundation: The Formation of a Microservices Ecosystem
MARCH 10, 2020

Bottlerocket – Open Source OS for Container Hosting
MARCH 10, 2020

SaltStack Enterprise and Salt Open
MARCH 10, 2020

The Case for Ephemeral Search
MARCH 10, 2020

Capture screenshots of flaky end-to-end tests
MARCH 10, 2020

Verifying upstream maintainers could help prevent supply-chain compromises
MARCH 10, 2020

UI5 Framework and UI5 Web Components
MARCH 10, 2020

Telecoms Tech News – Aether platform for 5G edge cloud as a service
MARCH 10, 2020

Performance Best Practices: Benchmarking
MARCH 10, 2020

Announcing new-look notifications for Slack
MARCH 10, 2020

Cloud Native Storage and Data Services with Diamanti
MARCH 09, 2020

CNCF Joins Google Summer of Code 2020!
MARCH 09, 2020

Stateful Cloud Native Applications - Why Reactive Matters
MARCH 09, 2020

Popular Python library, urllib3, subject to a denial of service vulnerability
MARCH 08, 2020

Microservices essentials: Getting started with Spring Cloud Gateway
MARCH 06, 2020

ASPEN MESH | Announcing Aspen Mesh 1.4.6 Security Update
MARCH 05, 2020

NS1. | Don't Let a DNS Outage Ruin Your Reputation
MARCH 05, 2020

portworx | Become a Portworx Certified Admin
MARCH 04, 2020

CiTRIX | Get greater visibility into your apps with Citrix ADM
MARCH 04, 2020

TRICENTIS | 5 Reasons Testing and Technology Leaders Shouldn't Miss Accelerate San Francisco 2020
MARCH 02, 2020

logdna | Custom Webhooks
FEBRUARY 11, 2020

packet | Building a Career in Tech
JANUARY 16, 2020

ARCHITECTURE	DEVELOPMENT	OPERATIONS	THE NEW STACK
Cloud Native	Cloud Services	CI/CD	Ebooks
Containers	Data	Culture	Podcasts
Edge/IoT	Development	DevOps	Events
Microservices	Machine Learning	Kubernetes	Newsletter
Networking	Security	Monitoring	About / Contact
Serverless		Service Mesh	Sponsors
Storage		Tools	Disclosures
			Contributions