ARM Caches: Giving you enough rope ... to shoot yourself in the foot

Marc Zyngier <marc.zyngier@arm.com>

KVM Forum '15

Caches on ARM: A technical issue? Or a cultural one?

```
From: Paolo Bonzini <pbonzini@redhat.com>
To: Christoffer Dall <christoffer.dall@linaro.org>

On 17/02/2015 18:54, Christoffer Dall wrote:
> yes, ARM is 'different' here

The correct spelling is "wrong". :)
```

KVM/ARM: *Lost in translation*

ARM

ARM and cache coherency

- What does ARM offer in terms of caches
- How are they architected
- How visible are they to software
- How and when to perform maintenance operations
- A few examples
- A few rants
- A way forward?

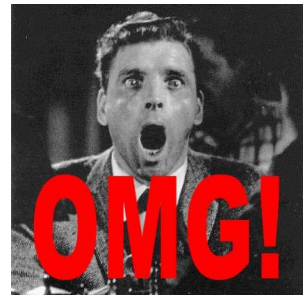
ARM

ARM: The cache coherency myth, and the facts

A common myth about the ARM architecture:

- The ARM architecture is not cache-coherent

Yeah, right.



ARM

ARM: The cache coherency myth, and the facts

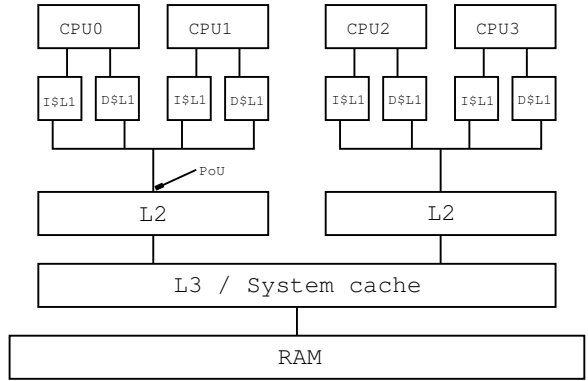
Now, the facts:

- Cache coherent architecture
- Scales from single CPU to massive SMP systems
- *Implementer* chooses to offer caches that are
  - visible to software
  - invisible to software
  - ... or any point between these two options
- Enough abstraction to cope with these differences
- Allows different PPA (*Performance, Power, Area*) points
  - Running a VM on your smart watch? Easy.
  - The same VM on your $15K server? Sure.
- The architecture is designed for maximum flexibility.

ARM

ARM: Cache architecture

- (Modified) Harvard architecture
  - Multiple levels of caching (with snooping)
  - Separate I-cache and D-cache (no snooping between I and D)
  - Either PIPT or non-aliasing VIPT for D-cache
  - Meeting at the *Point of Unification* (PoU)
- Controlled by attributes in the page tables
  - Memory type (normal, device)
  - Cacheability, Shareability
- Two Enable bits (I and C)
  - Actually not really an Enable switch
  - More like a global "attribute override"
- Generally invisible to *normal* software
  - With a few key exceptions...
  - More on that later



ARM: Interacting with caches

The ARM architecture offers the usual (mostly) privileged operations to interact with caches:

- Invalidate (I & D-cache)
- Clean (D-cache)
- Clean + Invalidate (D-cache)

- Cache maintenance by Virtual Address
- Cache maintenance by Set/Way

ARM: Interacting with caches

The ARM architecture offers the usual (mostly) privileged operations to interact with caches:

- Invalidate (I & D-cache)
- Clean (D-cache)
- Clean + Invalidate (D-cache)

- Cache maintenance by Virtual Address
- Cache maintenance by Set/Way

- Set/Way operations are local to a CPU
  - Will break if more than one CPU is active
- No *ALL* operation on the D side
  - Iteration over Sets/Ways
  - Only for bring-up/shutdown of a CPU
- Not all the levels have to implement Set/Way
  - System caches only know about VA
- Set/Way operations are impossible to virtualize

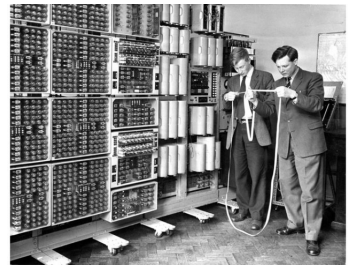
VA operations are the *only* way to perform cache maintenance outside of CPU bring-up/teardown

ARM

ARM: When caches become visible to software

Software needs to be aware of caches in a few cases:

- Executable code loading / generation
  - Requires a D-cache clean to PoU + I-cache invalidation
  - Possible from userspace on ARMv8
  - Requires a system call on ARMv7
- DMA with non cache-coherent devices
  - Requires the usual Clean, Invalidate, or both
- DMA with cache coherent devices when CPU caches are “off”
  - More surprising, but needs the same Clean + Invalidate
  - That’s because caches are never really off...
- Conflicting memory attributes
  - Writing to a non-cacheable mapping...
  - ... and expecting to read consistent data from a cacheable one.
  - Does it sound familiar?
  - This is where the proverbial rope kicks in



ARM

Introducing Stage-2 translation

Virtual machines add their share of complexity:

- Second stage of page tables (equivalent to EPT on x86)
- Second set of memory attributes
- KVM always configures RAM cacheable at Stage-2

These memory attributes get combined with those controlled by the guest:

- The *strongest* memory type wins
  - Device vs normal memory
- The *least cacheable* memory attribute wins
  - Non-cacheable is always enforced
- And the hypervisor doesn't have much control over it
  - Some global controls, but nothing fine grained

The noose is getting tighter.

A “benign” example

Booting a 32bit guest on a 64bit host (with an L3 system cache).

- The (compressed) kernel image is in RAM
- The embedded decompressor:
  - enables the caches,
  - decompresses the image
  - turns the cache off,
  - flushes it by *Set/Way*,
  - and jumps to the payload...

What could possibly go wrong?

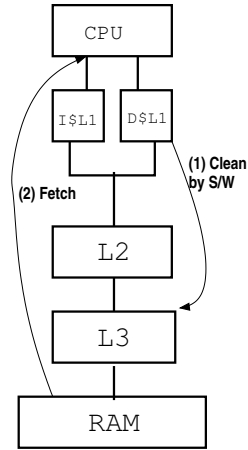
A “benign” example

Booting a 32bit guest on a 64bit host (with an L3 system cache).

- The (compressed) kernel image is in RAM
- The embedded decompressor:
  - enables the caches,
  - decompresses the image
  - turns the cache off,
  - flushes it by *Set/Way*,
  - and jumps to the payload...

What could possibly go wrong?

- System caches do not implement Set/Way ops
- So our guest code sits in L3, while fetching from RAM
  - We need to trap these ops and convert them into VA ops
  - Which means iterating over all the mapped pages
  - Good thing we’re only doing that at boot time!



ARM

A more annoying one

Let's imagine...

- A VM running on a (busy) host, swapping out pages
- A cache coherent I/O subsystem
- We have no visibility of the guest's memory attributes
- It could have written to memory from a non-cacheable mapping
- The page is swapped out via the host kernel's linear mapping

What could possibly go wrong *again*?

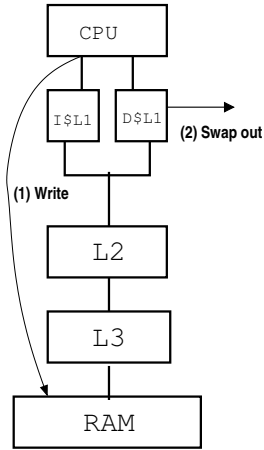
A more annoying one

Let's imagine...

- A VM running on a (busy) host, swapping out pages
- A cache coherent I/O subsystem
- We have no visibility of the guest's memory attributes
- It could have written to memory from a non-cacheable mapping
- The page is swapped out via the host kernel's linear mapping

What could possibly go wrong *again*?

- Requires an Clean + Invalidate on the page that is about to be evicted
- Could otherwise write out stale data (from the cached mapping)
- Always performed on Stage-2 unmap



ARM

What have we learned so far

- News flash: This is *NOT* the x86 behaviour
- Should that be surprising? See the logo at the bottom right...
- Caches are not just a "make it faster" block slapped on the side of the CPU
- They are an essential part of the coherency protocol
  - Using uncached memory explicitely bypasses it
  - It looks logical to cope with the consequences
- No magic involved!
  - Following the architecture rules ensures correctness on *all* implementations
  - No, Linux on 32bit is not architecturally compliant...

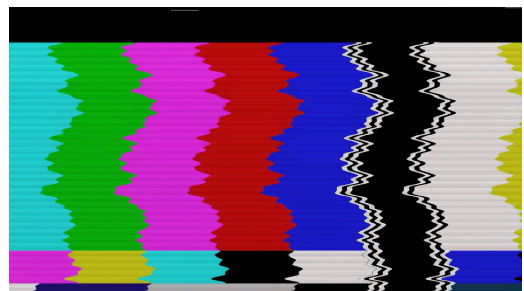
But of course, there is more to virtualization than just the CPU.

There is I/O...

<grumpy> Emulated devices: the uncached I/O issue </grumpy>

Top rant about KVM/ARM: *«My VGA adapter in QEMU doesn't work with KVM»*

- Userspace uses cached memory (via mmap)
- The guest uses non-cached memory
  - Why would the CPU read back from it?
- ... (you've noticed a pattern, haven't you?)
- Who needs a frame buffer anyway?



ARM

<grumpy> Emulated devices: the uncached I/O issue </grumpy>

Top rant about KVM/ARM: *«My VGA adapter in QEMU doesn't work with KVM»*

- Userspace uses cached memory (via mmap)
- The guest uses non-cached memory
  - Why would the CPU read back from it?
- ... (you've noticed a pattern, haven't you?)
- Who needs a frame buffer anyway?
- *«But it works with TCG!»*
- That doesn't make it more correct from an architectural PoV
- Something has to be done...

ARM

<grumpy> Emulated devices: the uncached I/O issue </grumpy>

How to fix this mess:

- Hack guest attributes, forcing cacheable
  - Breaks devices that *need* uncached access
- Cache maintenance from userspace
  - Requires a new syscall on ARMv7
- Allow userspace to mmap uncached
  - And what if the guest maps it as cached?
- Trap every `fscking` access
  - It will work, but...
- Just tell the guest the device is coherent
  - The only *real* solution
  - Lying to the guest is never good
  - Might require some surgery though

ARM

How did we end-up here?

A VGA device on an ARM VM looks like a terrible idea.

- VGA was invented in 1987...
- ... before ARM even existed as a company!
- ARM VMs have no legacy to care about
  - Hey, we don't even have (need?) a standard platform
- We use paravirtualized devices for most things
  - Console (of the byte stream persuasion)
  - Networking, storage...

How did we end-up here?

A VGA device on an ARM VM looks like a terrible idea.

- VGA was invented in 1987...
- ... before ARM even existed as a company!
- ARM VMs have no legacy to care about
  - Hey, we don't even have (need?) a standard platform
- We use paravirtualized devices for most things
  - Console (of the byte stream persuasion)
  - Networking, storage...
- Why don't we use virtio-vga as well?
  - We can make sure it is not encumbered by legacy
  - We don't have to lie about its virtual aspect
- Not *all* the world is an x86...
  - Our code base is riddled with assumptions

Emulated vs physical devices

Firmware does have some level of support to describe the cache coherency attributes:

- General need for topology information
  - This requirement exists on bare metal
  - VMs are no exception
- DT allows devices to specify their coherency
  - For PCI, this is set on a per-RC basis
- Mixing emulated devices (coherent) with physical devices
  - Physical devices may or may not be cache-coherent
  - May need separate RCs to be presented to the guest
- ACPI has its own set of attributes
  - _CCA: x86 mostly, but supported on ARM too
  - IORT: has support for all the ARM diversity

Though there seems to be a certain reluctance to expose them.

Conclusion

- KVM and its ecosystem are strongly x86 oriented (tainted?)
  - Not surprising, this is where it was created
- Not all the solutions that worked on x86 make sense on ARM
  - Nobody needs a Franken-VM
  - We have the chance of a clean slate
- It doesn't take much effort to fix KVM
  - All it takes is to follow the architecture requirements - to the letter
  - RTFAA (Read The Fabulous ARM ARM, almost 6000 pages - and counting)
- We already have modern, efficient solutions
  - Paravirtualization is the best thing since sliced bread
- Firmware (UEFI) seems to be the biggest issue
  - Probably the worse "x86-ism"
  - It isn't *that* hard to address the problems
  - Just don't assume that x86 is *the* solution

ARM

Thank You