arm

The art of virtualizing cache

Julien Grall <julien.grall@arm.com>

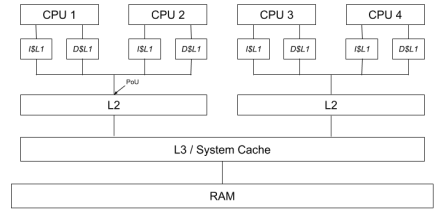
Xen Developer Summit 2018

Cache coherency on Arm

- 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.

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
  - An example is Executable code loading / generation



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

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

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
- Xen 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 much have control over it
  - Some global controls, but nothing fine grained

Linux 32-bit boot example

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

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

What could possibly go wrong?

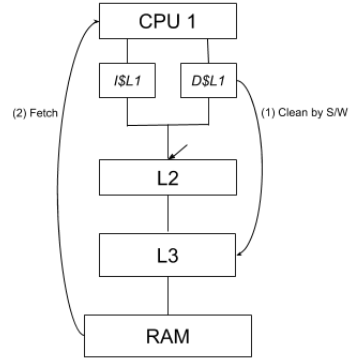
Linux 32-bit boot example

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

- The (compressed) kernel is in RAM
- The embedded decompressor:
  - enables the caches
  - decompress 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



arm

Set/Way in virtualized environment

The guest cannot directly use set/way because of:

- The presence of system caches on Arm64
- The vCPU can be migrated to another pCPU at any time
  - The new pCPU cache may not be cleaned

How can we solve this?

Set/Way in virtualized environment

The guest cannot directly use set/way because of:

- The presence of system caches on Arm64
- The vCPU can be migrated to another pCPU at any time
  - The new pCPU cache may not be cleaned

How can we solve this?

- 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!

Implementation of Set/Way in Xen

Xen and Set/Way today

- Set/Way instructions are not trapped
  - The guest is directly acting on the cache
- Potential cause of a heisenbug in Osstest
  - `https://lists.xenproject.org/archives/html/xen-devel/2017-09/msg03191.html`
- All guests using Set/Way are unsafe on Xen
  - Linux 32-bit
  - UEFI
  - ...

Cleaning guest memory

We need to iterate on each mapped page and clean them.

Any problems?

Cleaning guest memory

We need to iterate on each mapped page and clean them.

Any problems?

- Guest memory is always mapped
  - Lots of pages to clean
- 32-bit Linux is using Set/Way during CPU bring-up
  - Bring-up is bound by a timeout
- Pages are cleaned when first assigned to the guest

Cleaning guest memory

We need to iterate on each mapped page and clean them.

Any problems?

- Guest memory is always mapped
  - Lots of pages to clean
- 32-bit Linux is using Set/Way during CPU bring-up
  - Bring-up is bound by a timeout
- Pages are cleaned when first assigned to the guest

We need to clean only pages used since the last *flush*.

Trapping Set/Way instructions

Set/Way instructions usually happen:

- In batch of instructions
- Before turning on/off caches

A potential approach to trap would:

- On first Set/Way instruction
  - Enable trapping of VM instructions (e.g *HCR_EL2.TVM*)
  - Do a full clean of the guest memory
- Subsquent Set/Way instructions will be ignored until the cache is toggled
- On cache toggling
  - Do a full clean of the guest memory
  - Turn off trapping of VM instructions

Current status

- Some approach was discussed on Xen-devel in December 2017
  - `https://lists.xen.org/archives/html/xen-devel/2017-12/msg00328.html`
- A PoC based on the feedback was written
  - Sharing page-table is not possible with the approach
  - More details will be posted on *xen-devel*

Conclusion

- Caches are not just a ”make it faster” block slapped on the side of the CPU
- They are essential part of the coherency protocol
  - Using uncached memory explicitely bypasses it
  - It looks logical to cope with the consequence
- No magic involved!
  - Following the architecture rules ensures correctness on *all* implementations
  - RTFAA (Read The Fabulous ARM ARM, almost 7000 pages - and counting)

Questions?