Coherent Page Table updates

ARM

Julien Grall <julien.grall@arm.com>

Xen Developper Summit 2016

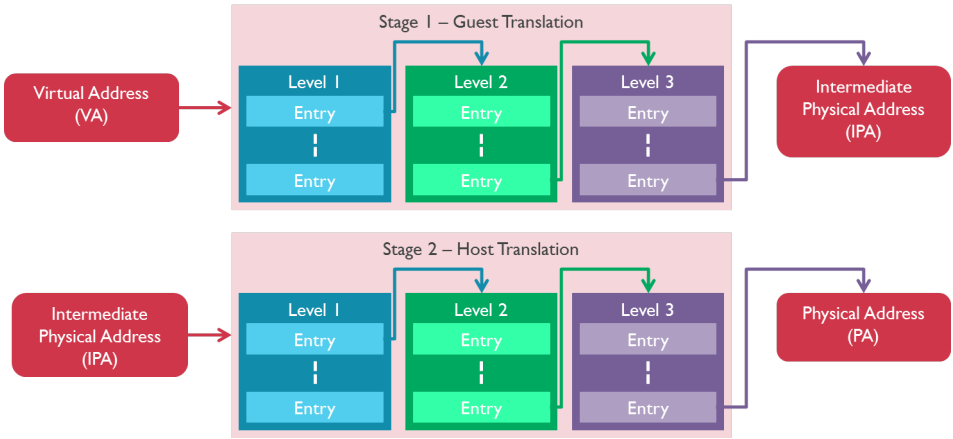
Scope

This talk is about handling page table updates on ARM.

- Focussing on architectural guarantees.
- The Architecture Reference Manual (”ARM ARM”) is authoritative.

Memory virtualisation

Hardware virtualisation of memory is achieved via two stages of page table.



Special considerations apply to translation table updates ...

ARM ARM

Page table updates

Page tables update may require to use a *break-before-make* sequence.

- It is a sequence required by the ARM ARM to update page table entries.
- Required for certain update of the tables.
- Ensures that a coherent view of the page tables is used by all observers.

Why do we care?

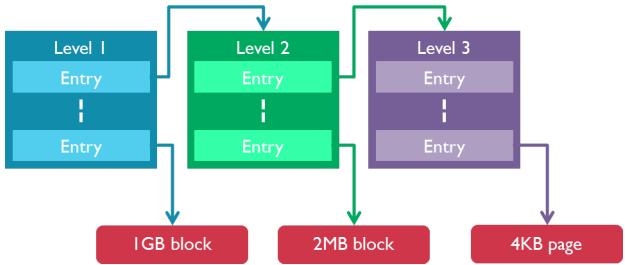
- Page tables are shared between CPUs.
- CPUs may still run during updates.

What could happen?

- TLB caching could break coherency.
  - A processor may see erroneous data or corrupt data.
  - This has been seen on various micro-architectures.
- A TLB conflict might be generated.
  - *IMPLEMENTATION DEFINED*
  - Either stage 1 or stage 2 abort

Page table entry

An entry can point to either a block mapping, translation table or a page descriptor.



Note that whilst ARM supports different granules (4KB, 16KB and 64KB), Xen only supports 4KB.

Page table entry - 2

Each block mapping and page descriptor contains information such as:

- Access permission: read, write, execute never
- Memory type: Device or Normal memory
- Shareability: non-shareable, inner-shareable, outer-shareable
- Cacheability
- Access flag
- Contiguous bit
- Non-Global bit

TLB entries

TLB is a structure which caches results of the translation table walks.

Each TLB entry

- contains data from the page descriptor.
- is tagged by ASID and VMID.

Contiguous bit

It indicates whether the entry is one of a number of adjacent page table entries that point to a *contiguous output address range*.

- The number of adjacent entries depends on the page size.
  - For instance with 4KB, it will be 16 entries.
- The TLB is allowed to cache a contiguous region in a single entry.

Address Space Identifier

The Address Space Identifier (ASID)

- identifies pages associated to a single address space (e.g. a process).
- applies for EL1 (kernel) and EL0 (userspace) translation regime.
- provides a mechanism for changing process-specific table without requiring TLB invalidation.
- is cached by the TLB entries.

Virtual Machine Identifier

The Virtual Machine Identifier (VMID)

- identifies the current virtual machine (Non-Secure EL1 & EL0).
- has its own independent ASID space.
- is cached by the TLB entries so no TLB invalidation is required when switching between VM.

Non-Global (nG) bit

- The non-Global bit applies only for Stage-1 page tables.
- *nG == 0* means the region is available for all ASIDs.
- *nG == 1* means the region is restricted to the current ASID.
- Global mapping can be invalidated by using any ASID.

Break-Before-Make

When to use it?

- Changing the size of the block
  - Replacing a block mapping with a translation table
  - Replacing a translation table with block mapping
  - Setting/Unsetting the contiguous bit
- Changing the output address if one of the entry is writeable
- Changing the memory type
- Changing the cacheability attributes
- Creating global entry that might overlaps non-global entries

When it is not necessary?

- Changing the permission of an entry
- Changing the access flag

TLBs are not allowed to hold any translation that generates a Translation Fault, an Address size Fault, or an Access Flag fault.

Steps

It is a 4 steps approach:

1. Replace the old entry with an invalid entry
2. Invalidate the cached old entries with a broadcasting TLB invalidation instruction
3. Wait for the completion of the TLB instruction with a dsb followed by an isb
4. Write the new entry

How does it ensure coherency?

The new and old entries will not be simultaneously visible to the other vCPUs.

- *Break*: Step 1 will remove the old entry
- Step 2-3 will ensure that all the processors see the update
- *Make*: Step 4 will make the new entry visible

Note that because the entry is invalid until step 4, the software may receive spurious fault. Spurious faults can be handled by some locking/serialization.

Xen and Break-Before-Make

Where?

Xen handles page tables in various places for

- the hypervisor page tables
- the stage-2 to protect each guest

Hypervisor memory layout - AArch32

	0x000000000
Xen text, data, bss	0x000200000
Fixmap: Special purpose 4KB mapping slots	0x000400000
Early boot mapping	0x000600000
	0x000a00000
Frame table	0x002000000
VMAP	0x008000000
Xenheap: always-mapped memory	0x040000000
Domheap: on-demand-mapped	0x080000000
	0x100000000

Xenheap vs Domheap

- Xenheap pages are always mapped in the virtual address space
  - Usually small (i.e at most 1GB on AArch32 by default)
  - Mostly used to allocate Xen internal memory
- Domheap pages are mapped on-demand
  - The code has to explicitely map/unmap the page when accessing it
  - On AArch32, the mapping is only done for the current CPU
  - Pages are often tied to a specific domain.
  - Used to allocate, for instance, guest RAM, stage-2 page table...

AArch64 has a large virtual address space (64 bits), so the two heaps are merged.

Hypervisor memory layout - AArch64

	0x0000000000000000
	0x0000000000200000
Xen text, data, bss	0x0000000000400000
Fixmap: Special purpose 4KB mapping slots	0x0000000000600000
Early boot mapping	0x000000000a00000
	0x0000000040000000
VMAP	0x0000000080000000
	0x0000000800000000
Frame table	0x0000001000000000
	0x0000800000000000
1:1 mapping of RAM	0x0000840000000000

Hypervisor page table handling

This has not yet been looked into details. Page tables are modified at runtime for

- Mapping a page from the domheap (only for AArch32).
  - per-CPU mapping
  - *map_domain_page*
- Mapping contiguously page in the virtual memory.
  - common for all CPUs
  - *create_xen_entries*

Stage-2 page table handling

- Stage-2 page tables are updated while the vCPUs are running
- *break-before-make* needs to be considered when
  - inserting a new mapping (the entry could hold a valid mapping)
  - Shattering a superpage
- *break-before-make* is not necessary for memaccess operations
  - memaccess is only changing the permission
  - however break-before-make is still required when shattering superpage

Upstreaming

An RFC has been posted on xen-devel

- https://lists.xen.org/archives/html/xen-devel/2016-07/msg02952.html
- Only handle stage-2 page tables
- Comments, reviews, testing are more than welcomed

ARM