Qualcomm

Scaling Hafnium for Advanced Mobile OS Architectures

Qualcomm Core Platform Team

Jack Suann

Agenda

- Requirements and Goals
- Proposals
- Memory Management Enhancements
- Questions

Our Requirements and Goals

- Support the use of highly scattered buffers.
  - Demanding memory usage requirements in Normal world OS means carveouts are not acceptable.
  - Standard memory allocation APIs in Linux and other OSes results in highly fragmented memory buffers.
  - DRM and other secure DMA use-cases require these buffers to be virtually contiguous.
- Optimizing memory overheads on memory constrained devices.
  - Limiting memory overhead of firmware is a strong customer requirement.
  - Complex use-cases add heap pressure and can easily result in OOM issues in an SPM.
- Give customers flexibility and scalability for their use-cases.
  - Support multiple SPs efficiently.
  - Support various DMA isolation model for peripheral devices.
  - Simplify implementations by avoid complex memory management within an SP.
- Contribute to Hafnium to help extend compliance with FF-A specification.
  - Without breaking any existing use-cases!

Extending Hafnium For Advanced Mobile OS

- Add support for non-identity memory mappings.
  - Enables virtually contiguous memory buffers in SPs and for secure DMA (without Stage-1 MMU).
  - Virtually contiguous buffers in SPs simplifies their implementation.
  - Contiguous IPA & VA regions reduces SPM's page table memory.
- Enable dynamic memory management for DMA devices.
  - Allow SPM to directly map FF-A buffers to be accessible by DMA.
  - Support more flexible control of DMA mappings from SPs.
- Support deployments with many multi-processor SPs.
  - Ensure Hafnium scales well on multi-core + multi-SP systems.
- Enhance deployment and debug capabilities.

Overview of Proposed Enhancements

Memory Management

- Non-identity mapping support
- Efficient normal world bookkeeping
- Multi-page RX/TX buffers
- Slab allocator

Scope of today's presentation

FF-A Proxy Support

- SEPID parsing in DT manifest
- SEPID support in FF-A memory management
- Shared stage-2 for SPs and DMA devices

SMMU Virtualization

- SMMUv3 stage-1 & nested support
- SMMUv3 fault handling
- Virtio-IOMMU for stage-1 management

Other Enhancements

- WCET analysis and improvements
- Debug facilities
- Misc optimizations

Memory Management

Support non-identity mappings in page table code

- Add non-identity variants of the mapping functions (prepare/commit) in the memory management code.
  - Keep existing identity mapping functions to maintain backwards compatibility.
- Challenges:
  - Some assumptions that input address == output address in page table code, such as during defragmentation.
- Status:
  - Prototype of changes complete.
- Impact:
  - With identity wrappers, impact is limited to page table code. Non-identity use-cases can be introduced as required.
  - Existing test cases pass, may require more extensive unit testing.

```
bool mm_vm_prepare(struct mm_ptable *ptable, ipaddr_t ip_begin, ipaddr_t ip_end,
                   paddr_t p_begin, mm_mode_t mode, struct mpool *ppool)
{
        struct mm_flags flags = mm_mode_to_flags(mode);

        return mm_ptable_prepare(
                ptable, ipa_addr(ip_begin), ipa_addr(ip_end), pa_addr(p_begin),
                arch_mm_mode_to_stage2_attrs(mode), flags, ppool);
}

bool mm_vm_identity_prepare(struct mm_ptable *ptable, paddr_t begin,
                            paddr_t end, mm_mode_t mode, struct mpool *ppool)
{
        return mm_vm_prepare(ptable, ipa_from_pa(begin), ipa_from_pa(end),
                             begin, mode, ppool);
}
```

Allow Borrowers to specify a composite memory descriptor

- Add support for handling composite memory descriptor in FFA_MEM_RETRIEVE_REQ.
  - Maintain existing IPA==PA as default behaviour when no descriptor is provided.
  - Composite memory descriptor allows SPs to control IPA / VA of where they want memory to be mapped.
- Challenges:
  - Requires additional bookkeeping in FF-A share states to track where memory is mapped for each receiver.
  - Need to ensure the SP cannot overwrite existing mappings (and memory states) in stage-2.
  - May require handling of fragmented retrievals – can we restrict the max number of requested regions to avoid this?
- Status:
  - Prototyping of base changes complete, with no support for fragmented retrieve requests.
- Impact:
  - Add checks and handling in FF-A memory retrieve.
  - Use new non-identity mapping page table interface.
  - Comprehensive testing of composite memory descriptor usage is required.

Support for non-identity donations?

- Supporting composite memory descriptors retrieval for donation adds new complexity.
  - SP owned memory may no longer be mapped 1:1.
- Challenges:
  - Need to remove Hafnium assumptions that owned memory is mapped 1:1.
  - SPs might try to use this memory for other purposes: FF-A shares, RX/TX buffers, etc.
    - FF-A memory donate needs to translate IPA to PA when parsing memory region descriptors.
    - Memory regions may not be physically contiguous, so FF-A share states need to support splintering the fragments.
- Status:
  - Prototyped an implementation supporting this.
  - Latest version rejects non-identity donation retrieval for now.
- Impact:
  - Extensive changes required to FF-A memory share states if owned memory can be non-identity mapped.
  - Can we live with 1:1 donation for now?

Alternative address allocation policies for FF-A retrieve requests

- Add support for non 1:1 relayer allocated addresses on retrieve.
  - Composite memory descriptors give SPs considerable flexibility in IPA / VA management.
    - This has a complexity cost for SPs that don't have special requirements for mapping memory.
  - Providing alternative retrieve policies and an address space allocator can support common SP requirements such as IPA allocation with virtually contiguous buffer retrieval.
- Challenges:
  - How should SPs choose which allocation policy to use?
    - Provide support for policy selection via the SP manifest.
    - Or specify a hint in FFA_RETRIEVE_REQ – will this require an extension to the FF-A memory management spec?
  - What policies should be implemented?
- Impact:
  - Requires an address allocation policy framework.
  - Requires new modules and data structures for address space management.

Efficient memory bookkeeping for “other world” VM

- Replace page tables of “other world” VM with an optimized bookkeeping structure.
  - Normal world ownership and state tracking is physical; no address translation is needed.
    - By only tracking the necessary page states we can reduce the memory usage of normal world bookkeeping.
  - Other benefits of not re-using the page table interface for this are:
    - Avoiding unnecessary TLBI and sync operations in the page table code.
    - Support more flexible data structures not limited by HW defined formats.
- Design:
  - Page tracking required: validity, ownership, sharing, normal / device attributes, access permissions.
  - Simplified page table-like structure, but entries at leaf levels are much smaller in size.
  - May require a fine-grained memory allocator for more sophisticated designs (structures smaller than 4KiB).
- Impact:
  - Requires new data structure for normal world page state bookkeeping.
    - This may also require a new internal memory allocator.
  - Only affects memory transactions involving normal world.
  - May result in performance improvements.

Multi-page RX/TX buffers

- NS OS and VMs may want larger RX/TX buffers for communication with SPs.
  - Larger than 4K RX/TX buffer support would allow for higher performance scattered buffer sharing.
    - Large memory descriptors requires transmission of descriptors in fragments, adding complexity and time overheads.
    - Fewer fragments reduces the number of world switches required when sharing from normal world.
  - Add support for negotiating larger RX/TX buffers with the normal world and SPs.
- Challenges:
  - Supporting non-contiguous buffers adds further complexity.
    - Limit support to only contiguous RX/TX buffers in Hafnium?
    - Hafnium could enforce that Normal world and SPs provide physically contiguous buffers to reduce complexity.
  - What is a reasonable upper limit for the max RX/TX buffer size?
    - We don’t want individual FF-A memory management operations to take too long in Hafnium either.
- Impact:
  - Limited direct impact on RX/TX map & unmap.
  - Need to address assumptions that mailbox size is always 4K.
    - Extensive changes required for removing mailbox size assumptions.

Fine-grained Memory Allocation

- Add a slab allocator for efficient fixed-size object allocations.
  - Support for smaller than 4K page-size allocations.
  - Suitable for kernels – reduces heap fragmentation, performance and efficient.
  - Potential uses:
    - FF-A memory descriptor entries, VCPUs, notifications.
    - Replacing static data structures: e.g. FF-A share states and VMs, improving scalability.
    - New data structures, such as efficient normal world memory bookkeeping proposal.
- Design:
  - Implement as a Safe Heap: storing metadata out-of-band for improved safety.
  - Could leverage the existing page pool allocator as a backend?
- Impact:
  - Depends on complexity of slabbing and how efficient we want allocations to be.
  - Could initially support power-of-2 size allocations, extending later to specific object sizes?

Possible Further Enhancements

- Tracking memory states (owner, shared, etc.) separately from page tables.
  - Moving these states outside of page tables has security / hardening benefits – can better prevent invalid state transitions.
  - Also avoids splitting blocks to pages when updating these memory states.
- Dis-contiguous RX/TX buffer support
  - Some NS VMs may request RX/TX buffers larger than 4K which are not physically contiguous.
  - Requires non-identity mappings in Hafnium EL2&0 address space to map these contiguously.
  - Could use top-half of EL2 stage-1 address space for these mappings?

Questions?

Thank you

Follow us on:

For more information, visit us at qualcomm.com & qualcomm.com/blog