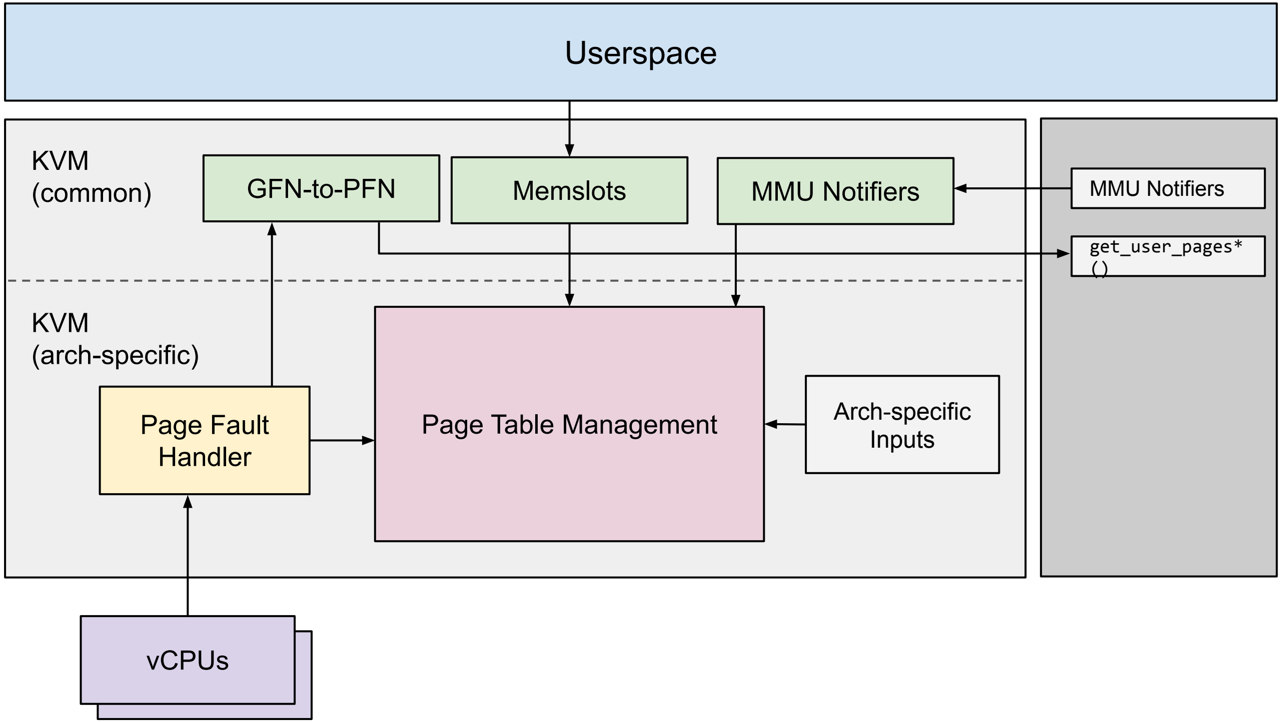
David Matlack
dmatlack@google.com

Exploring an architecture-neutral MMU

Google Cloud

KVM Memory Management from 10,000 ft



Google Cloud

KVM MMU in Cloud

- The scalability of the KVM MMU is important for Cloud use-cases.
- **Large VMs** with 100s of vCPUs and TiB of RAM.
- Customers with broad **range of workloads** and performance sensitivity.
- **Live Migration** part of host maintenance strategy.

KVM MMU in Cloud

- VMs use direct Two-Dimensional Paging (**TDP**).
  - Second stage of paging translates GPA to HPA.
- KVM/x86 can do shadow paging, but it's only required on ancient CPUs.
- KVM/ARM always requires TDP support (aka **Stage-2**)
- One exception: Nested Virtualization (more on this later later)

KVM MMU in Cloud

- Significant development has gone into making direct TDP on KVM/x86 scalable and performant for Cloud.
  - [2020] Introduction of the TDP MMU
  - [2021] TDP MMU Parallel Fault Handling
  - [2021] TDP MMU Lockless Write-Protection Fault Handling
  - [2022] TDP MMU Eager Page Splitting
- Further improvements are under active development.
  - NUMA-aware page table allocation
  - D-bit Dirty Logging
  - Multi-generational LRU support

Google Cloud

KVM/ARM MMU in Cloud

- Google Cloud recently announced T2A VMs, our first ARM-based VM offering.

Powered by Ampere® Altra® Arm-based processors, T2A VMs deliver exceptional single-threaded performance at a compelling price. Tau T2A VMs come in multiple predefined VM shapes, with up to **48 vCPUs per VM**, and **4GB of memory per vCPU**.

Google Cloud

KVM/ARM MMU in Cloud

- Lots of challenges to scale the KVM/ARM MMU for T2A VMs:
    - Interconnect scalability handling broadcast TLBIs and CMOs
    - ARM architecture makes certain PTE changes very expensive (break-before-make)
    - Over-aggressive TLB flushing in the KVM/ARM code
    - MMU Lock Contention
- There are many ARM-specific improvements address these.
    - e.g. local TLB flush instead of broadcast after resolving WP faults.
- But... we are also adopting techniques from the TDP MMU.
    - Parallelized Fault Handling to address MMU lock contention.
    - Eager Page Splitting to avoid huge page splitting faults.

Upcoming Live Migration Work

- Looking at our roadmap, there's a lot of overlap between x86 and ARM.

	KVM/x86		KVM/ARM	
Feature	**Want?**	**Have?**	**Want?**	**Have?**
Parallel Fault Handling	Yes	Yes	Yes	In-Progress
Eager Page Splitting	Yes	Yes	Yes	In-Progress
Lockless Write-Protection Fault Handling	Yes	Yes	Probably	No
D-Bit Dirty Logging	Probably	In-Progress	Probably	No
Multi-gen LRU support	Yes	In-Progress	Probably	No

- If other architectures (e.g. RISC-V) get traction in Cloud, we may have to maintain N copies of these features in KVM.

Google Cloud

Are there ways we can share code instead?

- The common theme among all these features is Page Table Management
- Plus a way to synchronize changes to page tables.
    - RW-Lock, RCU, atomic compare-exchange, and retries.

Feature	Page Table Operations
Fault Handling	Map @gfn to @pfn at @level
Eager Page Splitting	Split Huge Pages from @gfn_start to @gfn_end
Write-Protection Fault Handling	Relax Write-Protection Permission for @gfn
D-Bit Dirty Logging	Test and Clear PTE Dirty Bits in from @gfn_start to @gfn_end
Multi-gen LRU support	Test and Clear PTE Access Bits in from @gfn_start to @gfn_end

Making the TDP MMU Architecture-neutral

- Move the the TDP MMU from arch/x86/kvm/mmu/ to virt/kvm/mmu/.
- Delegate low-level implementation details to arch-specific code.
  - PTE bit layout, TLB flushing
- Expose an API for common page table operations:
  - Map @gfn to @pfn at @level
  - Relax Write-Protection Permission for @gfn
  - Split Huge Pages from @gfn_start to @gfn_end
  - Write-Protect from @gfn_start to @gfn_end
  - Test and Clear PTE Dirty/Access Bits in from @gfn_start to @gfn_end
  - Unmap from @gfn_start to @gfn_end
- Expose TDP MMU iterator for architecture-specific page table operations.

Making the TDP MMU architecture-neutral

Before*:

- 0% architecture-neutral
- 2295 LOC

```
180  arch/x86/kvm/mmu/tdp_iter.c
118  arch/x86/kvm/mmu/tdp_iter.h
1901 arch/x86/kvm/mmu/tdp_mmu.c
96   arch/x86/kvm/mmu/tdp_mmu.h
```

After**:

- 91% architecture-neutral
- 2653 LOC (+358 LOC)

```
1817 virt/kvm/mmu/tdp_mmu.c
179  virt/kvm/mmu/tdp_iter.c
250  include/linux/tdp_mmu.h
169  include/linux/tdp_iter.h

194  arch/x86/kvm/mmu/tdp_mmu.c
44   arch/x86/include/asm/tdp_pte.h
```

* At commit 90bde5bea810 (kvm/kvm-queue-post-20220525-rebase)
** RFC patches coming soon.

But can it support ARM?

- x86 and ARM both use a second stage of translation for VM memory.
    - Second stage both use a page table data structure.
- Page Tables are PAGE_SIZE.
- Page Table Entries are 64-bits.
- Page Table Entries can point to:
    - Page Tables
    - Huge Pages (aka Blocks)
    - Pages
    - Nothing (Invalid / Non-Preset)
- Page Table Entries can control Read/Write/Execute permissions as well as attributes.
    - e.g. cacheability

Differences between x86 and ARM

- x86: Total Store Ordering (TSO) memory model.
- ARM: Weakly ordered memory model.

Solvable? Yes.

- PTE writes need to use `smp_store_release()`
- Potentially some other minor changes.

Google Cloud

Differences between x86 and ARM

- x86: Pages are always 4KB
- ARM: Pages can be 4KB, 16KB, or 64KB

Solvable? Yes.

- KVM/ARM Stage-2 page size always follows Linux `PAGE_SIZE`.
- TDP MMU needs to key off of `PAGE_SIZE` when calculating e.g. number PTEs per table.
- TDP MMU levels need more abstract names. e.g.
  - `PG_LEVEL_4K` → `TDP_LEVEL_PTE`
  - `PG_LEVEL_2M` → `TDP_LEVEL_PMD`
  - `PG_LEVEL_1G` → `TDP_LEVEL_PUD`
  - etc.

Google Cloud

Differences between x86 and ARM

- x86: Root page table is always one page.
- ARM: Root page table can be concatenation of multiple page tables.
  - This can avoid a level of lookup, e.g. if root table would only use first N entries.

Solvable? Yes.

- Required for performance parity with KVM/ARM, but not correctness.
- Root page table allocator needs to be able to allocate contiguous page tables.
- TDP MMU iterator needs to be able to walk page tables with contiguous roots.

Differences between x86 and ARM

- x86: Huge Pages can be split in place.
  - i.e. replace huge PTE with a PTE pointing to a lower level page table.
- ARM: Software must use Break-Before-Make to split a huge page.
  - ... except if CPUs support FEAT_BBM=2.

Solvable? Yes.

- We can add Break-Before-Make to the TDP MMU Eager Page Splitting.
  - e.g. Behind `static_key` check for FEAT_BBM=2.
- Or just require FEAT_BBM=2 to use TDP MMU.

Note: Omitting Break-Before-Make can result in TLB conflict aborts, which can be expensive to handle (full local TLB invalidation). Periodic broadcast TLB invalidations during Eager Page Splitting would probably help.

Other Notable Differences

- ARM requires Break-Before-Make for certain PTE changes.
  - Solvable with unmap and let vCPUs fault back in. Eager Page Splitting is the only special case where avoiding faults is important for performance.
- ARM requires Cache Maintenance Operations (CMOs) after certain PTE changes.
  - Solvable with arch-hooks, but needs to be explored further.
- ARM does not guarantee Permission Faults evict or avoid creating TLB entries.
  - Solvable with local TLB invalidation after resolving permission faults on ARM.
- ARM allows combining contiguous PTEs to create intermediate huge page sizes.
  - KVM/ARM does not use Contiguous PTEs today in Stage 2.
    - i.e. *no need add support to the TDP MMU*
  - But, we may need reconsider if 16KB or 64KB granules gain traction for virtualization.

Google Cloud

pKVM

- TDP MMU not compatible with pKVM currently.
  - TDP MMU calls out to Linux (RCU, rescheduling, locking, allocation).
  - pKVM stage-2 page table management is done in the hyp; no access to Linux.
- TDP MMU could evolve to support pKVM.
  - Split out pure page table manipulation from higher level operations.
  - Use e.g. atomic counter + spinning instead of `call_rcu()` for page table freeing.
  - Opportunity to deploy pKVM to other architectures in a common way.
- Alternatively, pKVM could keep using separate Stage-2 code.
  - Android and Cloud are different use-cases.
  - But increases test and maintenance complexity.

Nested Virtualization

- The TDP MMU does not do shadow paging.
    - KVM/x86 uses separate code (shadow MMU) for nested.
    - KVM/ARM would need to do the same.
- Architecture-neutral shadow paging for nested?
    - Difficult, since shadowing is inherently more architecture-specific.
        - Guest hypervisor could use any architectural feature.
    - Paravirtualization could be a path toward architecture-neutral nested support.
- Note: the TDP MMU *does* interoperate with shadow paging.
    - Write-protecting guest page tables + software tag bit in PTE.
    - Hooks for handling write-faults to guest page tables.

Conclusion

- About 90% of the existing TDP MMU can be made architecture-neutral.
- Using the TDP MMU for ARM Stage-2 is feasible, but comes with caveats.
  - pKVM would not be supported initially.
  - CPUs with FEAT_BBM < 2 can't use TDP MMU (optional code-complexity trade-off)
- RFC patches to split the TDP MMU into architecture-neutral code is coming soon.
- Open Question: Would any other architectures be interested? RISC-V?