KVM/ARM at Scale

Improvements to the MMU in the face of hardware growing pains

12 September 2022

Oliver Upton, Google

Google Cloud

Introduction

Google Cloud has recently announced the T2A family of VMs, the first product built with the Arm architecture

- Ampere Altra SoC
- KVM-based virtualization stack
- Close-to-upstream “Icebreaker” kernel (presented at OSS 2021)

Dirty Tracking on Arm

- MMU lock contention was the bounding issue
- Write protection is the name of the game
    - No feature like Intel's PML
- High frequency of stage-2 aborts
- Dirty state tracked at PTE granularity

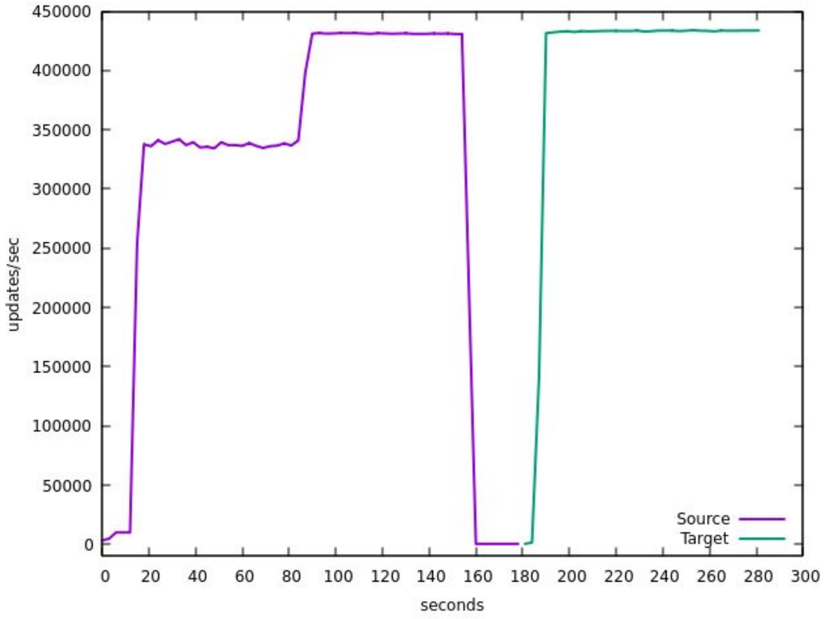
Test Workload

Theoretical worst case scenario:

- t2a-standard-48 (48 vCPUs, 192 GiB RAM)
- Backed with 2M HugeTLB
- Guest userspace strides memory with 100% write accesses
- After some time VMM enables dirty logging

“Live” Migration

- >99% performance degradation when dirty logging is enabled
- Guest starved of CPU for nearly 30 seconds



Google Cloud

Here we go again...

At first glance, the problems are similar to x86:

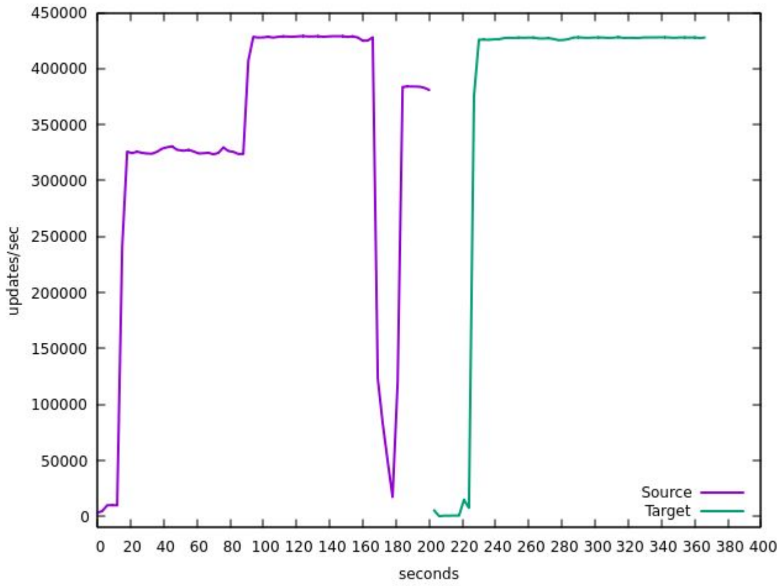
- MMU guarded with a spinlock
- When dirty logging, blocks are split into tables lazily

We went about fixing the problem the same way:

- 5.18: Take the read lock to write-unprotect a page
- RFC: Take the read lock for the other stage-2 faults

Signs of life

- Improvement over baseline
- Still, significant performance degradation at the beginning of dirty logging



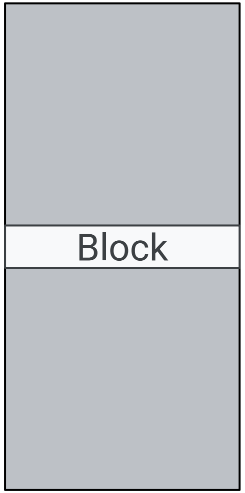
Where else are we serializing?

- Inspecting some traces, it appears a lot of time is spent in `__kvm_tlb_flush_vmid_ipa()`
- Called in the middle of page split because of break-before-make
- No software locking, so what gives?

Break-before-make

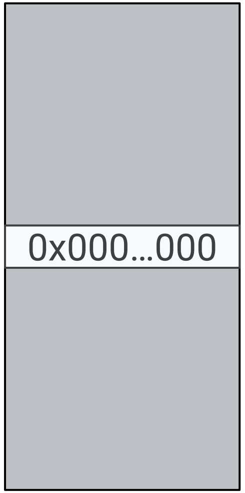
- Arm architecture more prescriptive than others (x86) on how software manipulates page tables
- Software must first make an invalid PTE (break) visible to hardware before another valid PTE (make)
- Prevents TLB conflicts
- Required for hugepage splitting

Break-before-make (cont’d)



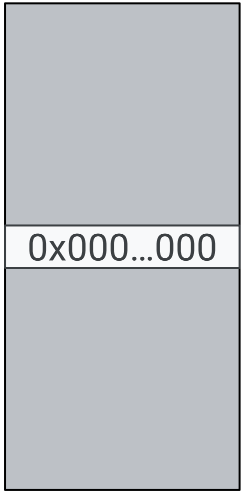
```
WRITE_ONCE(*ptep, 0);

dsb(ishst);

tlbi(ipas2e1is, ipa);

dsb(ish);

tlbi(vmalle1is);

dsb(ish);

isb();

WRITE_ONCE(*ptep, new);
```

Break-before-make (cont’d)


0x000…000

```
WRITE_ONCE(*ptep, 0);

dsb(ishst);

tlbi(ipas2e1is, ipa);

dsb(ish);

tlbi(vmalle1is);

dsb(ish);

isb();

WRITE_ONCE(*ptep, new);
```

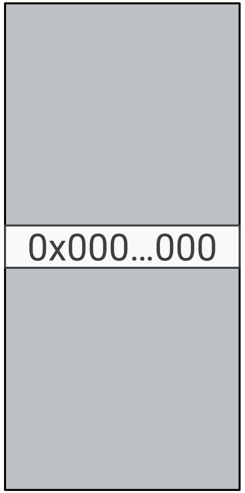
Break-before-make (cont’d)



0x000...000

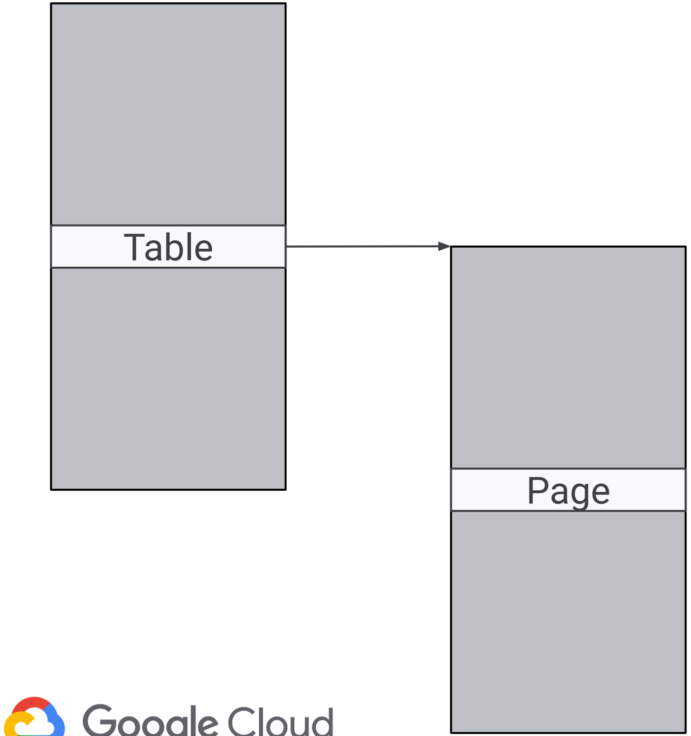
```
WRITE_ONCE(*ptep, 0);

dsb(ishst);

tlbi(ipas2e1is, ipa);

dsb(ish);

tlbi(vmalle1is);

dsb(ish);

isb();

WRITE_ONCE(*ptep, new);
```

Break-before-make (cont'd)



```
WRITE_ONCE(*ptep, 0);

dsb(ishst);

tlbi(ipas2e1is, ipa);

dsb(ish);

tlbi(vmalle1is);

dsb(ish);

isb();

WRITE_ONCE(*ptep, new);
```

Break-before-make (cont’d)



```
WRITE_ONCE(*ptep, 0);

dsb(ishst);

tlbi(ipas2e1is, ipa);

dsb(ish);

tlbi(vmalle1is);

dsb(ish);

isb();

WRITE_ONCE(*ptep, new);
```

Google Cloud

Side effects of break-before-make

- TLB invalidations are broadcasted to Inner-Shareable domain
- DSB awaits the completion of *all* in flight invalidations on the Inner-Shareable domain
- Observation: on a loaded system, the sequence can take several **milliseconds** to complete
- Result: unacceptable vCPU fault latency

What if I elide break-before-make?

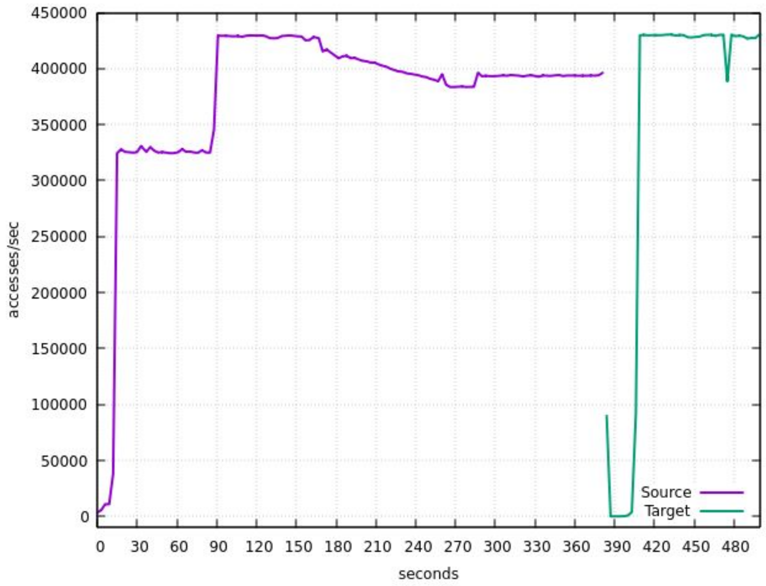
- Based on the implementation:
  - TLB conflict abort
  - TLB returns either of the duplicate entries
  - TLB returns an amalgamation of both entries
- Open season for all kinds of interesting failures, such as breaking:
  - Coherency
  - Single-copy atomicity
  - Ordering

Mitigating in software

- Eliminate unnecessary broadcasting of TLB invalidations
  - Relaxing write permissions falls outside the scope of break-before-make
  - Instead, invalidate only within the Non-Shareable domain (local)
- Spread out the necessary TLB invalidations over a longer period of time
- Solution: extend the KVM_CLEAR_DIRTY_LOG ioctl to split hugepages
  - Eager page splitting, with the ability to ratelimit in userspace

End result

- Page splitting throttled to minimize break-before-make overhead
- Gradual (and smaller) degradation in guest performance



Outlook

- Problem only gets worse with more cores in a system
  - Interconnect implementations need TLB snoop filters
- `FEAT_TLBIRANGE` - Software can target a range of memory with a single invalidation; allows batching without dropping all context
- `FEAT_BBM=2` - Relaxes the break-before-make requirements, allowing hugepage split/collapse without the sequence
  - Snag: software needs to deal with TLB conflict aborts. Only option is to flush everything when the abort occurs.

Acknowledgements

- Ricardo Koller: Eager splitting implementation
- David Matlack: Use CLEAR ioctl for split throttling
- Marc Zyngier: Non-Shareable TLB invalidations
- Jing Zhang: Parallel permission relaxation

Questions?

Google Cloud