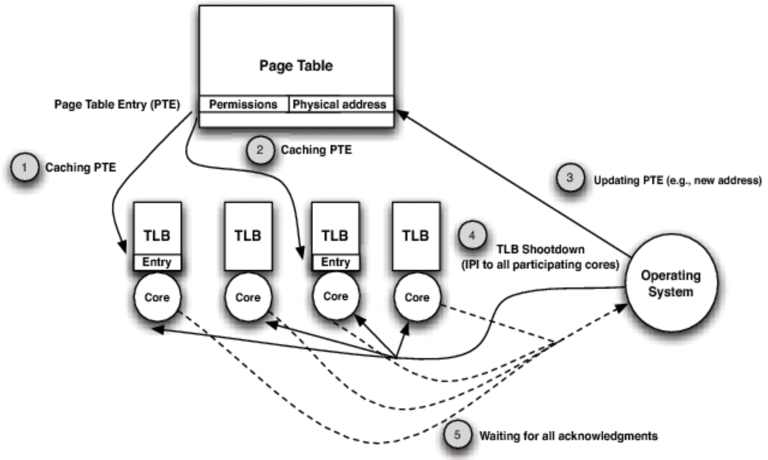
Linaro Connect
MADRID 2024 | MAY 14-17 2024

Why TLBI matters on ARM server: scalability issues we found and solutions

Hanjun Guo <guohanjun@huawei.com>
Xiang Chen <chenxiang66@hisilicon.com>

What is TLBI?

- TLB
  - ✓ Translation Lookaside Buffer.
  - ✓ It's a cache (translation cache), caching the mappings of VA to PA, critical for the performance.
- TLBI
  - ✓ TLB is cache, but without full coherent, so any update to the shared page table, software needs to "maintain the coherence", which is TLB invalidation.
  - ✓ Anything related to page table attribute and mapping update, a TLBI will be needed.
  - ✓ Usecases such as VM live migration, Tired memory, ARM contiguous bit and more...

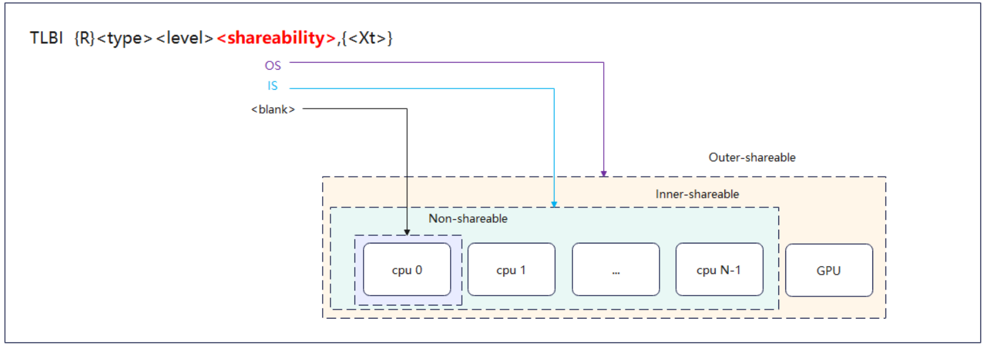


TLBI broadcasting

Every TLB invalidation operation uses the following template:

```
DSB ISHST     // Ensure prior page-table updates have completed
TLBI ...      // Invalidate the TLB
DSB ISH       // Ensure the TLB invalidation has completed
```

From arm ARM， it said that TLBI will broadcast to IS （Inner shareable） or OS （Outer shareable） ，inner shareable and out sharable are IMPLEMENTATION DEFINED, but usually IS or OS will include all the CPUs in the system.



For the typical inner shareable system:

- ✓ TLBI will broadcast to all the CPUs
- ✓ DSB ISH just wait TLBI to be retired

TLBI is time consuming on multi-core system

Why?

Because as defined by the AMBA CHI spec, only one DVM(Distributed Virtual Memory) Sync can be executed at one time on the CHI bus, with CPU number increased, the waiting list of DVM Sync on MN(Misc Node) will be increased, leading to the delays.

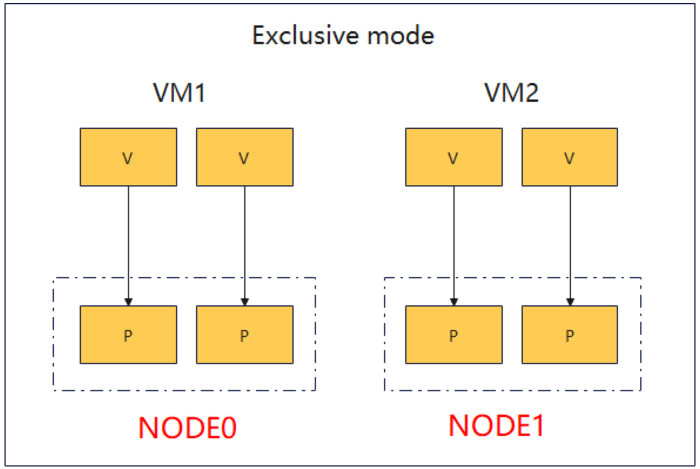
And NUMA, latency is added for multi CPU Die and sockets, which adding more time consuming.

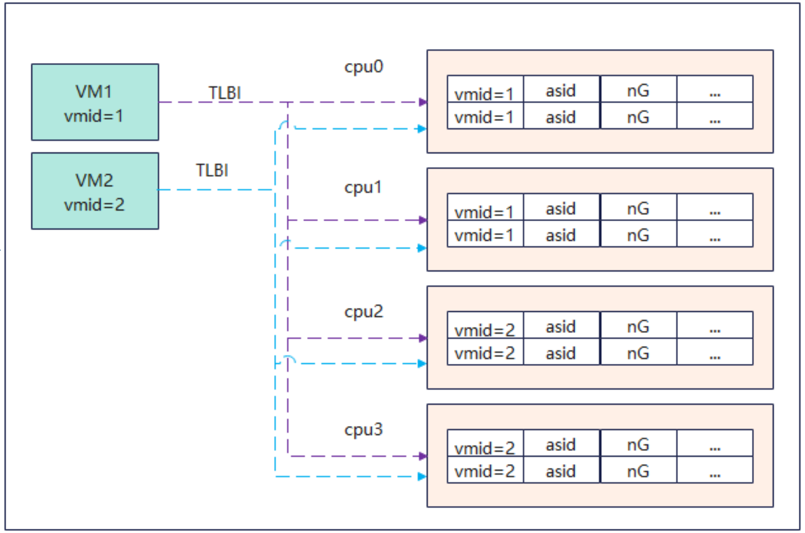
For virtual machine use cases, HCR_EL2.FB is set in default, DVM request from guest OS will broadcast to the whole inner sharable domain.

All of this will introduce scalability issues on a multi-core system, especially on ARM servers(lots of them are around 100 cores).

TLBI scalability issues running VMs (1)

Virtual machines will only run on some specific CPUs in the system, even the CPU is shared by multi VMs, but whenever shared translation tables are modified, TLBI will send to all the CPUs in the system.





vmid=1	asid	nG	...
vmid=1	asid	nG	...

vmid=1	asid	nG	...
vmid=1	asid	nG	...

vmid=2	asid	nG	...
vmid=2	asid	nG	...

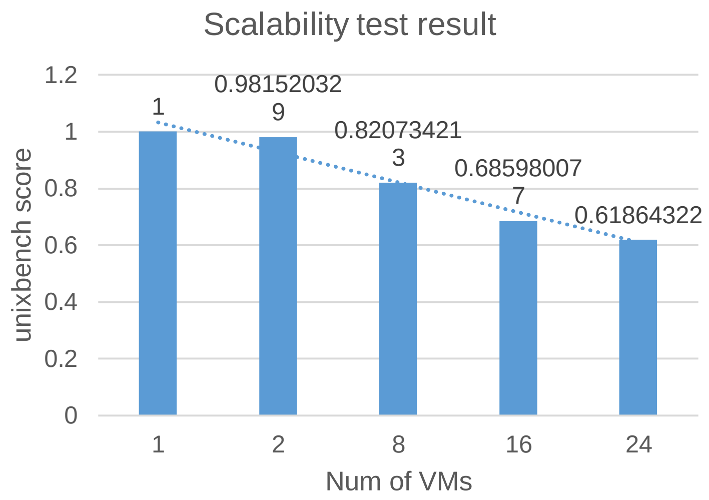
vmid=2	asid	nG	...
vmid=2	asid	nG	...

TLBI scalability issues running VMs (2)

On a 96 cores ARM server, bootup 1/2/8/16/24 VMs， each VM with 4 CPU cores, running unixbench in the VM to test the performance, which is:

- ✓ Round 1, bootup only one VM with 4 CPU cores on a 96 core system, running unixbench;
- ✓ Round 2, bootup two VMs, each VM with 4 CPU cores, running unixbench;
- ✓ …
- ✓ Round 5, bootup 24 VMs…

It doesn't scale well from 1 to 24 VMs, TLBI broadcast is like neighbor noise.



*The data of unixbench scores is normalized to (0,1)

Previous work for mitigating the TLBI overhead

Reduce TLBI times (accepted by mainline)

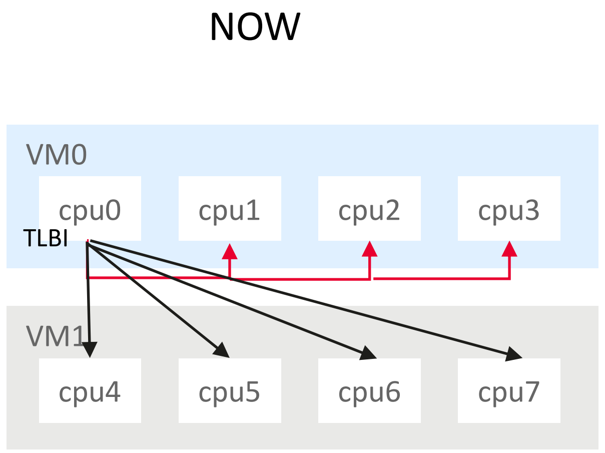
1. Avoid synchronous TLB invalidation for intermediate page-table entries on arm64, by Will Deacon [here]
2. TLBI by range for ARMv8.4， by Zhenyu Ye [here]
3. TLBI by range for KVM, by Raghavendra [here]
4. Batched/deferred tlb shootdown, by Yicong Yang, Barry Song and Anshuman Khandual [here]

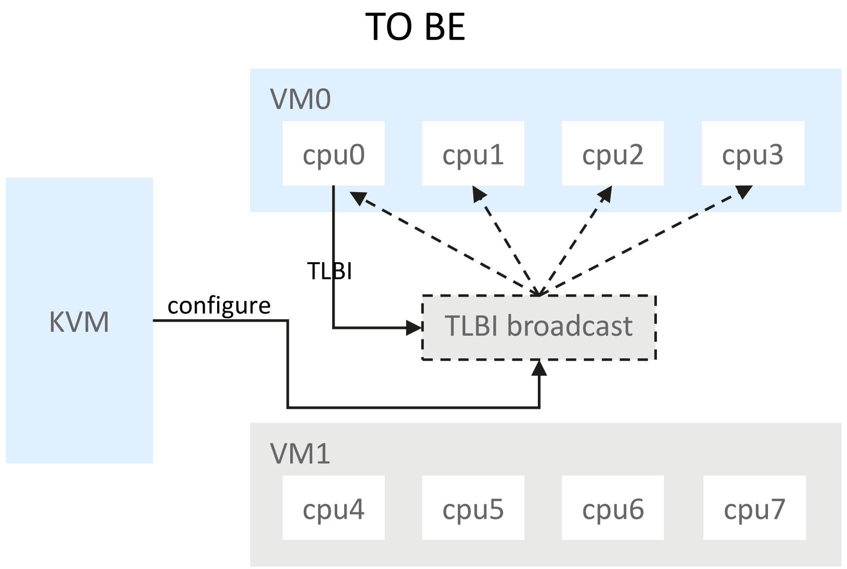
Limit TLBI broadcast scope (not accepted)

1. IPI based TLB invalidation, by Matthias Brugger 8 years ago [here]

Still don’t have a good way to limit the TLBI broadcast scope.

TLBI by Affinity: Software-Hardware co-design for TLBI overhead optimization





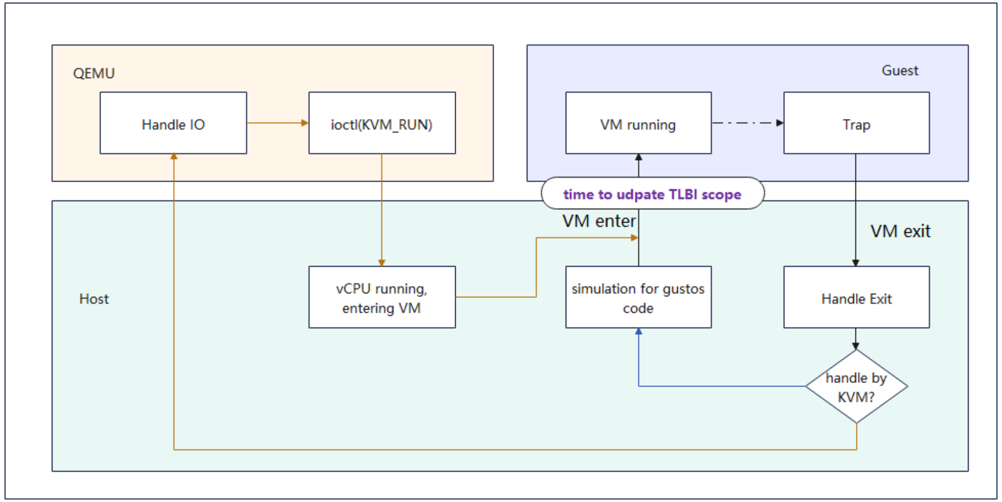
Allows TLBI executed at EL1 to be broadcast in a configurable range of physical CPUs (even with HCR_EL2.FB set)

TLBI by Affinity: how it works for the hardware

1. For the CPU running VM, introduce a per CPU core register, to record the TLBI broadcast affinity for which CPUs will run, the software update it accordingly.
2. Some microarchitecture level updates, such as DVM broadcast.

TLBI by Affinity: how it works for the software

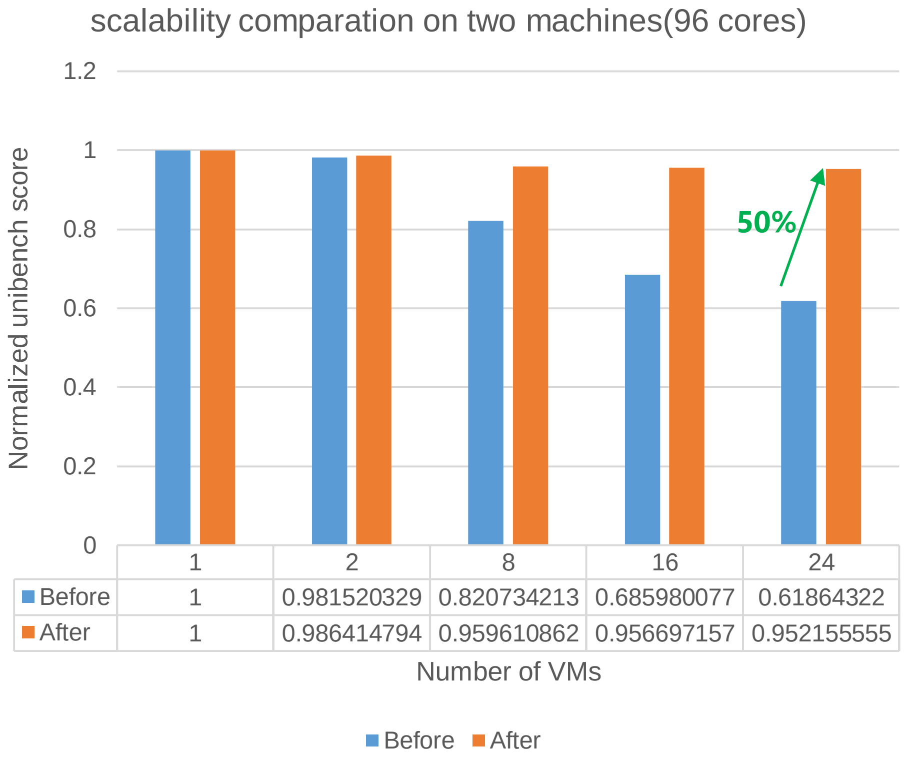
1. KVM needs to record and update the physical CPU bitmap of the running VM in real time;
2. Before each vcpu is loaded, we re-calculate the VM-wide sched_cpus, and if it's changed we will kick all other vcpus out to reload the latest TLBI affinity to the register, otherwise keep it unchanged.



TLBI by Affinity: Performance data

Do the same test on two machines with 96 cores (bootup 1/2/8/16/24 VMs running unixbench), with the TLBI by affinity:

1. Scale well from 1 to 24 VMs.
2. More performance gain with CPU core added.
3. Compared to TLBI without optimization, we got 50% performance boost for 24 VMs.



	1	2	8	16	24
🟦 Before	1	0.981520329	0.820734213	0.685980077	0.61864322
🟧 After	1	0.986414794	0.959610862	0.956697157	0.952155555

TODO

- Extend the TLBI by affinity to per process/thread granularity.
- “Upstream” to the ARM spec, then upstream to the mainline kernel.

Linaro Connect

MADRID 2024 | MAY 12-17 2024

Thank you