Improving RISC-V TLB Shootdown Performance

John Henry Deppe deppe@ece.ubc.ca
Guy Lemieux lemieux@ece.ubc.ca
Department of Electrical and Computer Engineering, University of British Columbia

Abstract

Popular instruction set architectures have recently added new instructions to assist TLB shootdown, a software task performed by the OS to ensure that writes to the page table are seen by all address translation caches. RISC-V currently uses a slower traditional approach where TLB invalidation is done by broadcasting an inter-processor interrupt (IPI) to all affected harts. The IPI sender must stall until it collects acknowledgments from all target harts; each target hart is normally running some other software but it must take an exception to run a trap handler that evicts the specified TLB entry or entries and sends back an acknowledgment. In this early-stage research, we are reviewing the state-of-the-art in hardware and software techniques to accelerate TLB shootdown, and instrumenting the Linux kernel to help develop a new RISC-V mechanism that can accelerate remote TLB invalidations. The most promising techniques in research use a familiar hardware feature: cache coherence.

Introduction

This work describes early research that will propose ways to accelerate TLB shootdowns in RISC-V.

Virtual Memory (VM) is an extremely important feature that isolates programs from each other. It improves security and reliability; memory being used by one program can't be seen by other programs, and a bug in one program won't corrupt other programs.

While VM is powerful, performance in multi-core systems suffers from a particular inter-process interference called "TLB shootdown" which is necessary to prevent harts from using stale virtual-to-physical address translations [1, 2]. Initiated by the OS on one hart for certain page table changes, TLB shootdown is broadcast via inter-processor interrupt (IPI) to all affected harts; each of those harts temporarily stops running its process and flushes the modified page table entries (PTE) from its address translation cache(s), also known as translation lookaside buffer(s) (TLB). Operating systems research has improved the frequency (how often), scope (number of targeted harts), and performance impact of TLB shootdowns.

Despite these efforts, the OS still sends more interrupts than necessary because it lacks precise information about which translations are present in each hart. In this work, we are instrumenting the Linux kernel on a RISC-V processor system to measure exactly how many harts are unnecessarily included in TLB shootdowns. From that information, we will design new hardware-level mechanisms enabling the OS to filter more harts from shootdown or elide IPIs entirely.

Hardware Prior Art

A naive system flushes the TLB on every context switch. Modern architectures, including RISC-V, avoid this with address space identifiers (ASIDs). Each TLB entry is tagged by the ASID that fetched it and is used only by that ASID. This permits TLB entries to safely persist through context switches for reuse when their ASID resumes. Using ASIDs increases the scope of shootdowns, but shootdowns for inactive ASIDs can be deferred to the next context switch.

AMD, ARM and Intel have instructions to avoid costly IPIs. AMD's INVLPGB invalidates a range of addresses in both local and remote TLBs. Early Linux results with INVLPGB show good promise [3]. ARM's TLBI is similar to INVLPGB, but performance issues slowed adoption. Intel's Remote Action Request (RAR) bundles remote invalidation commands in shared memory, but is not yet in Linux.

In research, DiDi [4] uses an on-chip shared directory to track which CPU cores share a PTE mapping. Upon shootdown, the directory non-intrusively invalidates all shared copies of the mapping. Both UNITD [5] and HATRIC [6] use cache coherence instead of special directory hardware. Upon writing to a PTE, any sharing harts receive a cache coherence transaction. These transactions search the TLB using extra hardware (a CAM and tag bits), and more invalidations than necessary are performed. To eliminate these overheads, ATTC [7] memory maps a last-level TLB into DRAM and allows the data cache to hold these address translations. None of these research systems have been built into a physical system.

Software Prior Art

Deferring and batching TLB shootdowns reduces interruptions by removing several entries with one interrupt or context switch [8]. Reduced synchronization results in less waiting for IPI acknowledgment, and overlapping waiting with local invalidations improves concurrency [9]. Additional deferred invalidation opportunities exist, but their safety is controversial [10].

The current RISC-V Linux kernel uses ASIDs and sends TLB shootdowns only to harts that have executed the process in question. When possible, interrupts are avoided by deferring shootdown to context-switch time. Yet, this is still overspecified because (1) any given page is unlikely to have been touched by all of these harts, and (2) based on age, remaining harts may or may not have evicted the mapping. Better precision and granularity is needed — this may demand new architectural support to expose fine-grained information about which harts are sharing address translations.

This Work

An open question remains: What should RISC-V do? Add remote invalidation like AMD, ARM, and Intel? Exploit cache coherence instead? Can a new approach be developed?

Other architectures' remote invalidation instructions show us that today's shootdown IPI overheads can be greatly reduced. However, many-core systems may also benefit from shootdowns being addressed to only essential harts, and this suggests a tracking mechanism similar to cache coherence.

To start, we are instrumenting the Linux kernel for fine-grained, per-page tracking of which harts have actually loaded an address translation. Although we expect significant slowdown, it will generate essential data with real applications that will help us design future hardware-assisted mechanisms.

Inspired by ABIS's tracking each page using a single bit to designate local (not shared) or global (shared) [11], we use a bitmask to track exactly which harts have loaded each translation. To reduce this substantial overhead, we only track pages that have been explicitly marked with the `madvise()` syscall.

Applications that unmap or reduce page permissions across multiple cores, such as larger-than-memory databases, incur TLB shootdown overhead. Even if only one hart used and cached a given translation, current RISC-V systems may shootdown all harts the process has ever run on to invalidate that translation. These spurious shootdowns are reclaimable overhead.

Instrumentation Details

Our RISC-V system uses a hardware page table walker. To track each hart that loads a translation, we mark user application's `madvise()`'d PTEs as *invalid*. Then a page fault occurs when a hart first accesses the page, allowing us to use custom instructions to manually load the correct address translation into the TLB *and* to add the faulting hart to the sharing bitmask of that mapping. Further accesses proceed normally. With this, the OS can track a translation's TLB residence on a per-process, per-hart, and per-page basis.

The operating system is not immediately informed of TLB evictions, but this does not cause incorrectness. After a tracked page is evicted, an additional fault reloads the TLB and informs the OS of the eviction.

Now that the OS knows which harts hold an address translation, it can send shootdown interrupts to only those harts that have actually accessed and possibly cached that translation. This improves over the current RISC-V Linux state-of-the art where shootdowns may be sent to all harts that have run the process. This most benefits multi-threaded applications where few harts have used the translation being shot down.

Future Work

The software mechanism described above incurs page fault overheads and may only be useful for collecting data. With that data, we can best determine whether it is valuable enough to add new hardware-based tracking, cache coherence-based TLB invalidation, or hardware-assisted broadcast TLB invalidation similar to AMD, ARM, and Intel's instructions. The synchronization-reducing Svinval and Svvptc extensions will be considered and appropriately integrated.

Acknowledgements

This work received funding from Andes Technology. We thank Steven Yeung for his help and support.

References

[1] D. L. Black et al. "Translation Lookaside Buffer Consistency: A Software Approach". In: *ASPLOS*. Apr. 1989.

[2] P. J. Teller. "The Cost of TLB Consistency". In: *Cache and Interconnect Architectures in Multiprocessors*. 1990.

[3] M. Larabel. *Benchmarking the AMD INVLPGB Linux Kernel Patches for Better Performance*. Phoronix. 2024.

[4] Carlos Villavieja et al. "DiDi: Mitigating the Performance Impact of TLB Shootdowns Using a Shared TLB Directory". In: *PACT*. Oct. 2011.

[5] Bogdan F. Romanescu et al. "UNified Instruction/Translation/Data (UNITD) Coherence: One Protocol to Rule Them All". In: *HPCA*. Jan. 2010.

[6] Zi Yan et al. "Hardware Translation Coherence for Virtualized Systems". In: *ISCA*. June 2017.

[7] Harsh Gugale et al. "ATTC (@C): Addressable-TLB Based Translation Coherence". In: *PACT*. Sept. 2020.

[8] Volkmar Uhlig. "Scalability of Microkernel-Based Systems". PhD thesis. Universität Karlsruhe (TH), 2005.

[9] Nadav Amit, Amy Tai, and Michael Wei. "Don't Shoot down TLB Shootdowns!" In: *EuroSys*. Apr. 2020.

[10] M. K. Kumar et al. "LATR: Lazy Translation Coherence". In: *ASPLOS*. Mar. 2018.

[11] Nadav Amit. "Optimizing the TLB Shootdown Algorithm with Page Access Tracking". In: *USENIX ATC*. 2017.