LATR: Lazy Translation Coherence

Mohan Kumar[*], Steffen Maass[*], Sanidhya Kashyap, Ján Veselý[‡], Zi Yan[‡], Taesoo Kim, Abhishek Bhattacharjee[‡], Tushar Krishna

Georgia Institute of Technology [‡]Rutgers University

* Co-First Authors

March 28, 2018

Motivation

Supermicro Debuts 8-Socket Server for Intel Xeon Processors

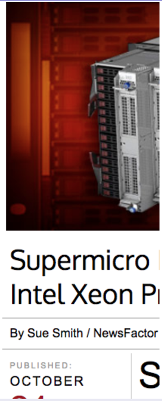
By Sue Smith / NewsFactor Network

PUBLISHED:
OCTOBER

Supermicro just announced the latest addition to its line of SuperServer systems, designed for data centers and
e SuperServer 7089P-TR4T is an
ver for Intel Xeon scalable

Large NUMA machines

Motivation



Supermicro

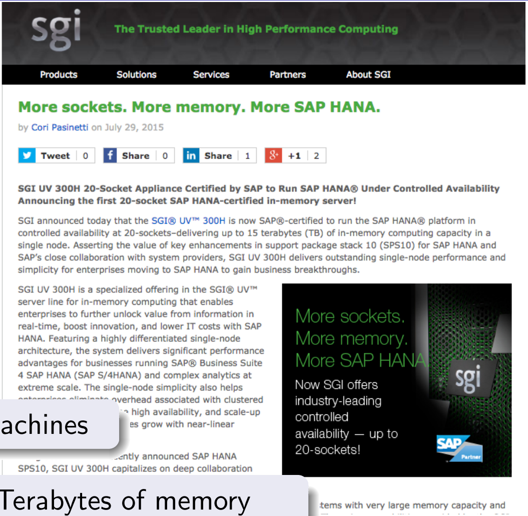
Intel Xeon P

By Sue Smith / NewsFactor

PUBLISHED:

OCTOBER

S



More sockets. More memory. More SAP HANA.

by Cori Pasinetti on July 29, 2015

SGI UV 300H 20-Socket Appliance Certified by SAP to Run SAP HANA® Under Controlled Availability
Announcing the first 20-socket SAP HANA-certified in-memory server!

SGI announced today that the SGI® UV™ 300H is now SAP®-certified to run the SAP HANA® platform in controlled availability at 20-sockets–delivering up to 15 terabytes (TB) of in-memory computing capacity in a single node. Asserting the value of key enhancements in support package stack 10 (SPS10) for SAP HANA and SAP's close collaboration with system providers, SGI UV 300H delivers outstanding single-node performance and simplicity for enterprises moving to SAP HANA to gain business breakthroughs.

SGI UV 300H is a specialized offering in the SGI® UV™ server line for in-memory computing that enables enterprises to further unlock value from information in real-time, boost innovation, and lower IT costs with SAP HANA. Featuring a highly differentiated single-node architecture, the system delivers significant performance advantages for businesses running SAP® Business Suite 4 SAP HANA (SAP S/4HANA) and complex analytics at extreme scale. The single-node simplicity also helps enterprises eliminate overhead associated with clustered

high availability, and scale-up

es grow with near-linear

ently announced SAP HANA SPS10, SGI UV 300H capitalizes on deep collaboration

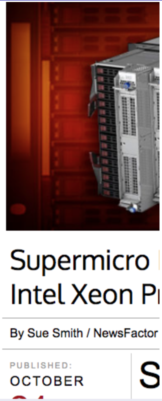
tems with very large memory capacity and

More sockets.
More memory.
More SAP HANA.
Now SGI offers industry-leading controlled availability — up to 20-sockets!
sgi
SAP Partner

Large NUMA machines

Terabytes of memory

Motivation



Supermicro
Intel Xeon Pr

By Sue Smith / NewsFactor

PUBLISHED:
OCTOBER

S

More sockets. More m

by Cori Pasinetti on July 29, 2015

SGI UV 300H 20-Socket Appliance
Announcing the first 20-socket SA

SGI announced today that the SGI® U
controlled availability at 20-sockets–de
single node. Asserting the value of key
SAP's close collaboration with system
simplicity for enterprises moving to SA

SGI UV 300H is a specialized offering i
server line for in-memory computing t
enterprises to further unlock value fro
real-time, boost innovation, and lower
HANA. Featuring a highly differentiated
architecture, the system delivers signi
advantages for businesses running SA
4 SAP HANA (SAP S/4HANA) and comp
extreme scale. The single-node simplic
enterprises eliminate overhead associa
high availab
es grow with

ently announce
SPS10, SGI UV 300H capitalizes on de

Large NUMA machines

Microsecond-scale I/O means tension between performance and productivity that will need new latency-mitigating ideas, including in hardware.

BY LUIZ BARROSO, MIKE MARTY, DAVID PATTERSON, AND PARTHASARATHY RANGANATHAN

Attack of the Killer Microseconds

THE COMPUTER SYSTEMS we use today make it easy for programmers to mitigate event latencies in the
d millisecond time scales (such as
at tens or hundreds of nanoseconds
and disk I/Os at a few milliseconds) but significantly
le events. This

Terabytes of memory

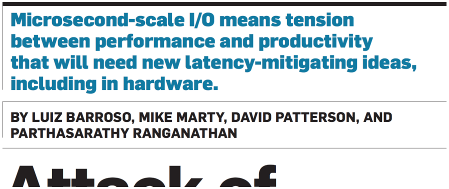
Microsecond latency

Motivation

More sockets. More n

by Cori Pasinetti on July 29, 2015

SGI UV 300H 20-Socket Appliance

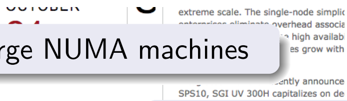


Microsecond-scale I/O means tension between performance and productivity that will need new latency-mitigating ideas, including in hardware.

BY LUIZ BARROSO, MIKE MARTY, DAVID PATTERSON, AND PARTHASARATHY RANGANATHAN

Attack of

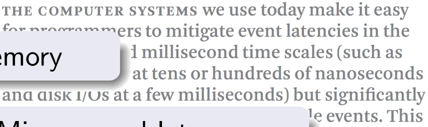
⇒ Problem of Microsecond Latency in System Services
⇒ TLB Coherence is Contributor in Important Subset



OCTOBER

extreme scale. The single-node simplic
enterprises eliminate overhead associa
high availab
es grow with
ently announce
SPS10, SGI UV 300H capitalizes on de

Large NUMA machines



THE COMPUTER SYSTEMS we use today make it easy for programmers to mitigate event latencies in the
d millisecond time scales (such as
at tens or hundreds of nanoseconds
and disk I/Os at a few milliseconds) but significantly
le events. This

Terabytes of memory

Microsecond latency

Impact of TLB coherence on applications

- Multi-core MapReduce application
  - Prior research: **10x increase in shootdown time** with increasing core counts
- Web servers (e.g., Apache)
  - Prior research and our findings: ≈**35% of time spent in TLB shootdown**
- Die-stacked Memory
  - Swapping between on-chip and off-chip memory
- Disaggregated Memory
  - Swapping between local and remote memory

Impact of TLB coherence on applications

- Multi-core MapReduce application
  - Prior research: **10x increase in shootdown time** with increasing core counts
- Web servers (e.g., Apache)
  - Prior research and our findings: $\approx$**35% of time spent in TLB shootdown**
- Die-stacked Memory
  - Swapping between on-chip and off-chip memory
- Disaggregated Memory
  - Swapping between local and remote memory

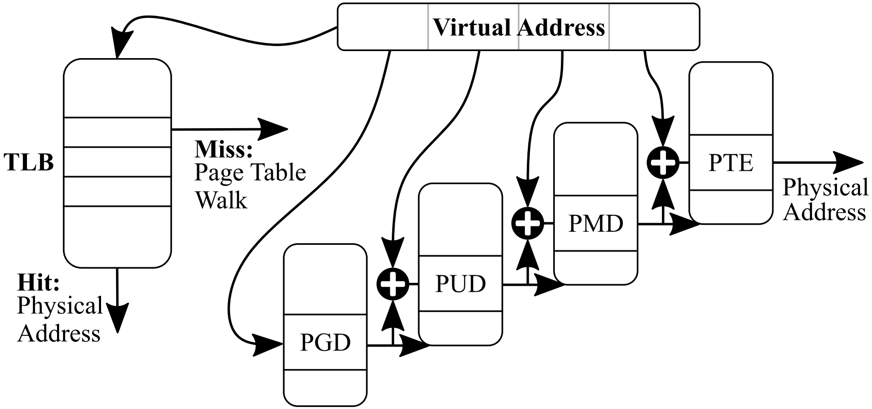
$\Rightarrow$ Can we mitigate this costly TLB shootdown?

Table of contents

Table of contents

Translation lookaside buffer: Introduction

- Cache for virtual → physical mapping, per-core structures
- Accessed on every load/store
- Unlike data caches (L3, etc.), coherence managed by OS
- TLB coherence significantly impacts application performance



TLB coherence: Background

- **Hardware-based Approaches**
  - Providing cache coherence to TLBs
  - ISA-level instruction support (ARM)
  - Microcode-based approaches
- **Software-based Approaches**
  - Current commodity OS design: Use **Inter-Processor Interrupts (IPI)**
  - Optimization: Reduce number of shootdowns, better tracking
  - Multikernel design: Use Message-Passing

TLB coherence: Background

- **Hardware-based Approaches**
  - Providing cache coherence to TLBs

⇒ More Hardware Complexity

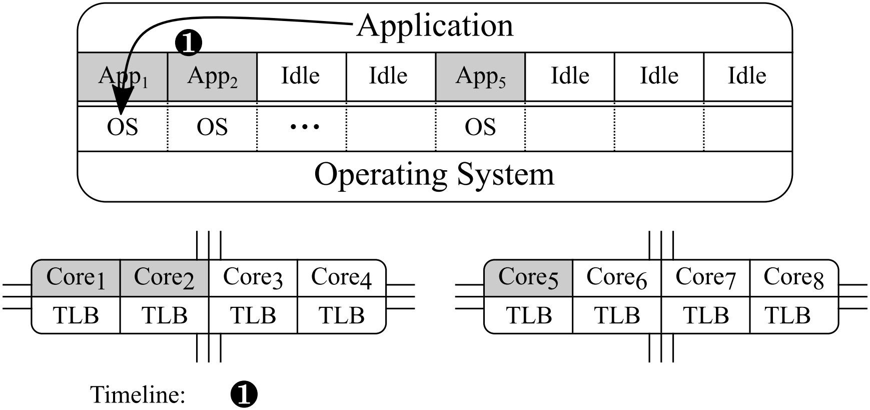
- **Software-based Approaches**

⇒ TLB shootdowns still significant

  - Optimization: Reduce number of shootdowns, better tracking
  - Multikernel design: Use Message-Passing

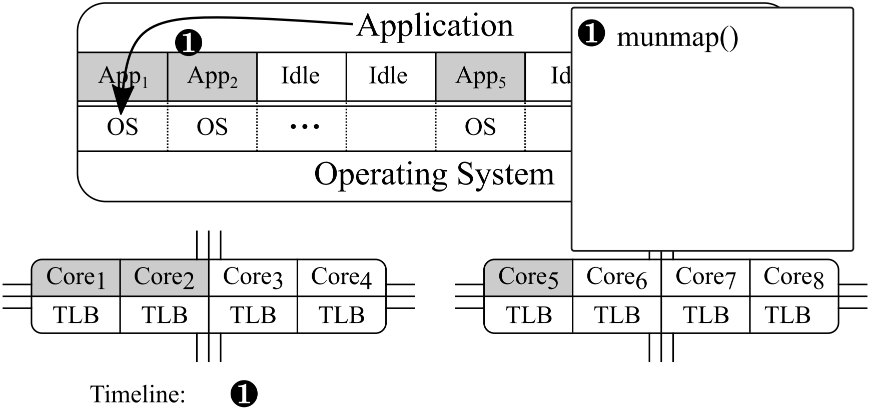
TLB shootdown internals in Linux

- `munmap()` on core 1, application running on cores 1, 2, and 5:



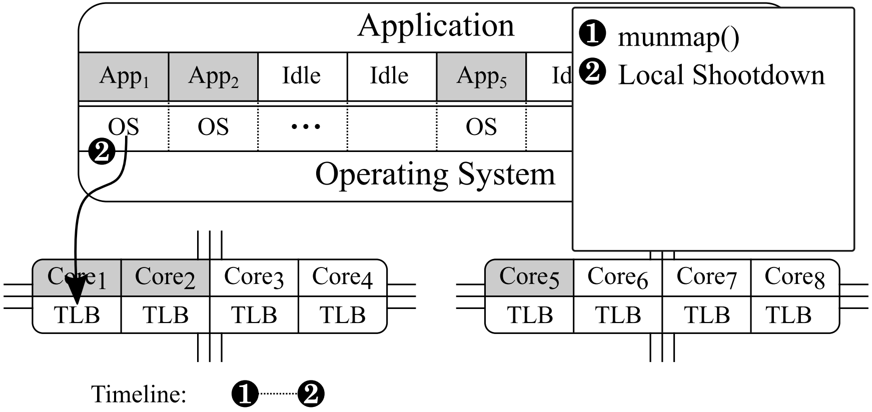
TLB shootdown internals in Linux

- `munmap()` on core 1, application running on cores 1, 2, and 5:



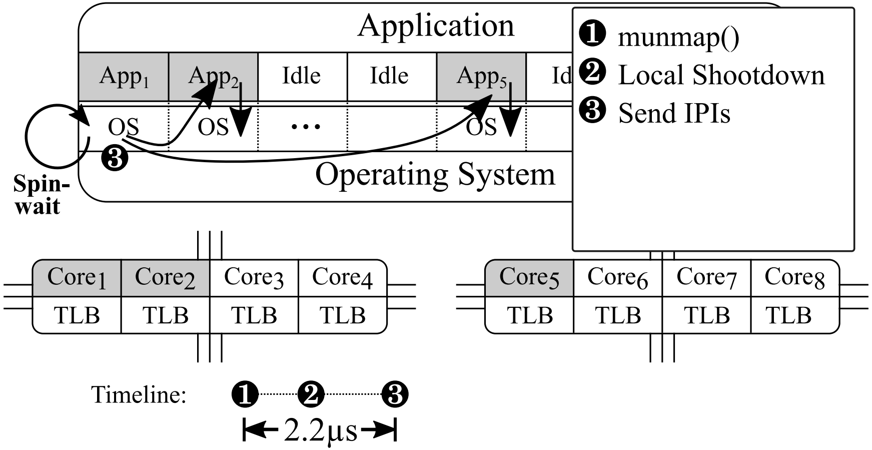
TLB shootdown internals in Linux

- Context switch on core 1, local TLB shootdown:



TLB shootdown internals in Linux

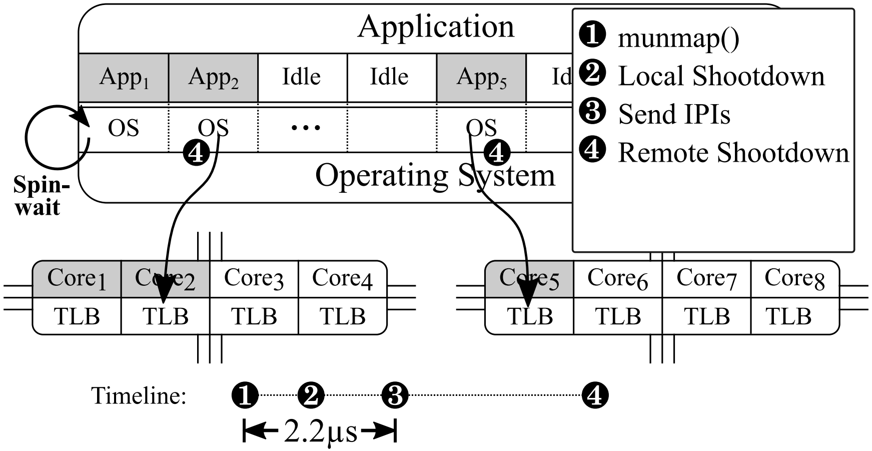
- Notify cores 2 and 5 via IPI, application blocked on core 1:



❶ munmap()
❷ Local Shootdown
❸ Send IPIs

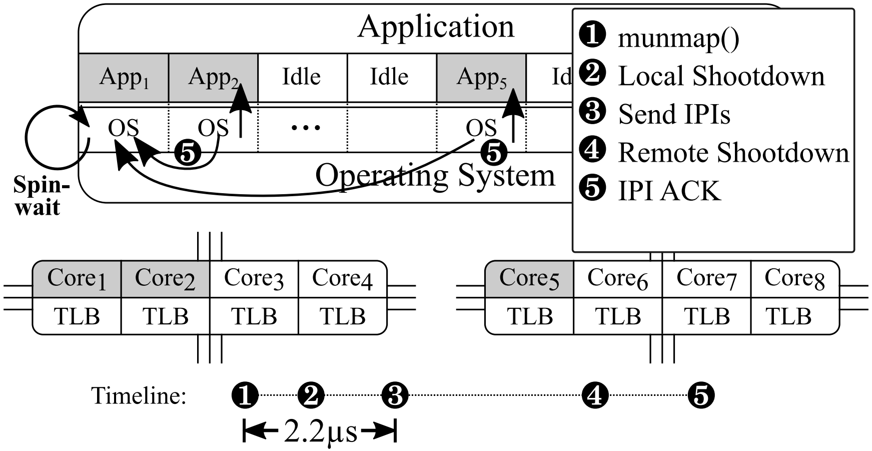
TLB shootdown internals in Linux

- Execute context switch and TLB shootdown on cores 2 and 5:



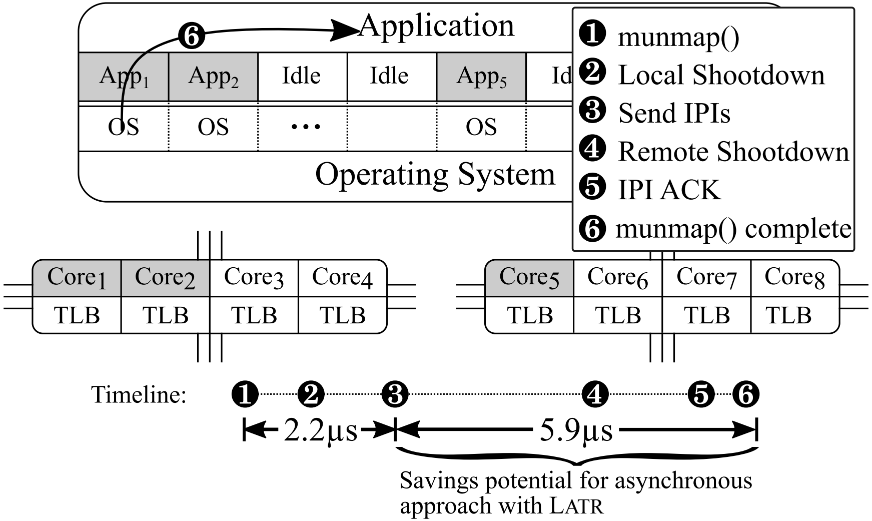
TLB shootdown internals in Linux

- Cores 2 and 5 respond ACK via shared memory:



TLB shootdown internals in Linux

- Control is returned on all cores, TLB shootdown completed:

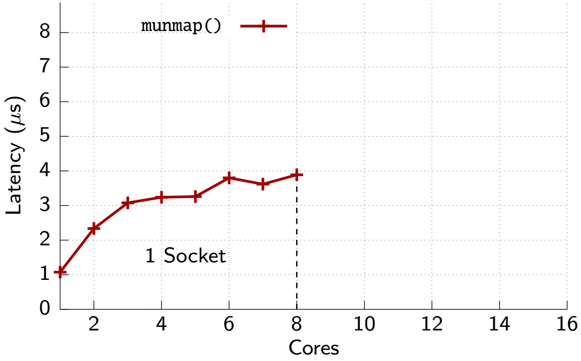


Observation

- **Synchronous TLB shootdown is expensive:**
  - Up to 6 μs delay with two sockets
- **Processing IPIs is expensive:**
  - Interrupt handler on remote core
  - Long wait time on initiating core
- **IPI send-and-wait delay:**
  - Unicast delivery of the IPIs (one at a time)

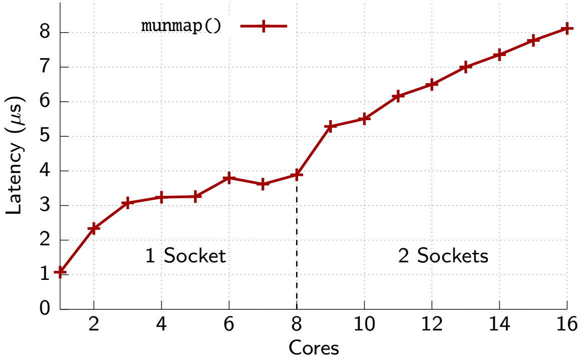
TLB shootdown: A necessary evil

- Cost of a simple memory unmap operation (`munmap()`):
  - 1 page on 16 cores with 2 sockets: **up to 8 μs**
  - $\approx$ 70% **from TLB shootdown alone**
- More expensive with more sockets:



TLB shootdown: A necessary evil

- Cost of a simple memory unmap operation (`munmap()`):
  - 1 page on 16 cores with 2 sockets: **up to 8 μs**
  - $\approx$ 70% **from TLB shootdown alone**
- More expensive with more sockets:



TLB shootdown: A necessary evil

- Cost of a simple memory unmap operation (`munmap()`):
  - 1 page on 16 cores with 2 sockets: **up to 8 μs**
  - $\approx$ 70% **from TLB shootdown alone**
- More expensive with more sockets:

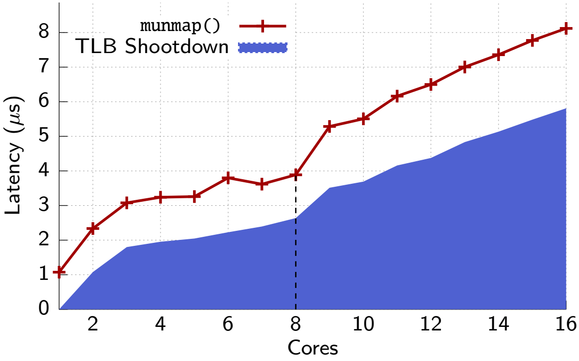


Table of contents

In this talk: LATR

- LATR: **La**zy **Tr**anslation Coherence
- **Perform asynchronous TLB shootdown**
  - Remove remote shootdown from the critical path
  - Take advantage of change in ABI without affecting applications' correctness
- **Use shared memory instead of IPI**
  - Eliminate send-and-wait delay of IPIs
- **Scope:**
  - *free* operations (in this talk)
  - *migration* operations (see our paper)

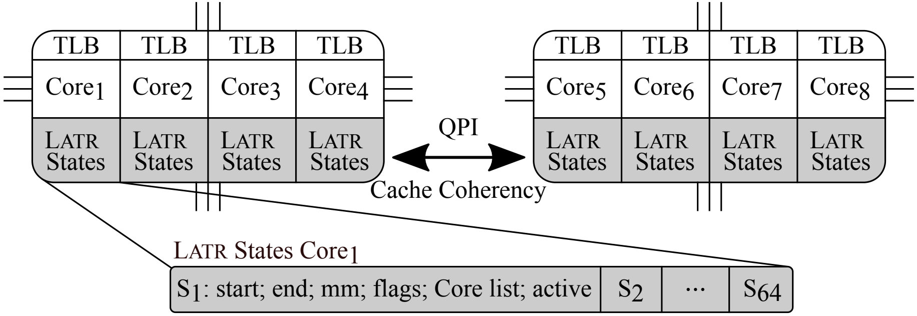
In this talk: LATR

- LATR: **La**zy **Tr**anslation Coherence
- **Perform asynchronous TLB shootdown**
  - Remove remote shootdown from the critical path
  - Take advantage of change in ABI without affecting applications' correctness
- **Use shared memory instead of IPI**
  - Eliminate send-and-wait delay of IPIs
- **Scope:**
  - *free* operations (in this talk)
  - *migration* operations (see our paper)

$\Rightarrow$ But: How to perform asynchronous shootdown?

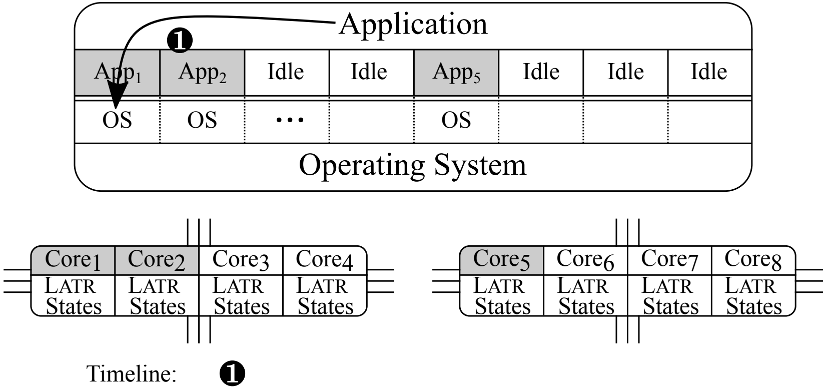
LATR States

- Store virtual addresses to be flushed
- Remote cores shootdown local TLB during
  - OS context switch
  - OS scheduler tick (**upper bound**: 1ms in Linux)



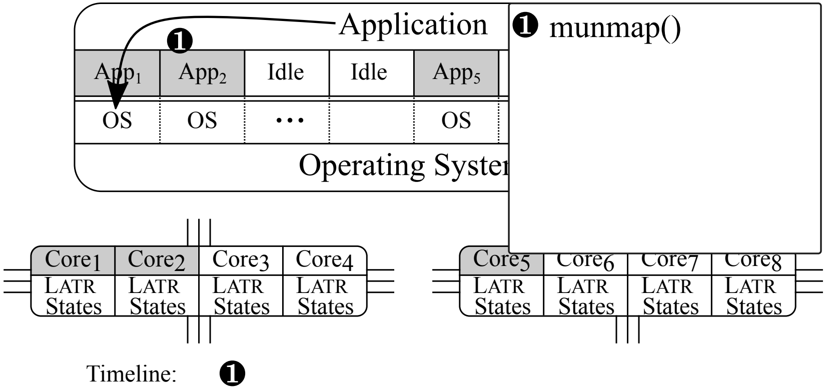
LATR: Example

- munmap() initiated on core 1:



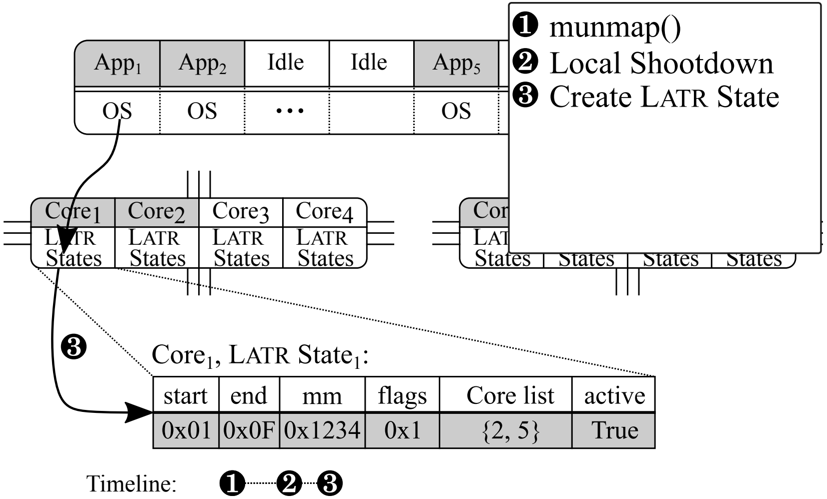
LATR: Example

- `munmap()` initiated on core 1:



LATR: Example

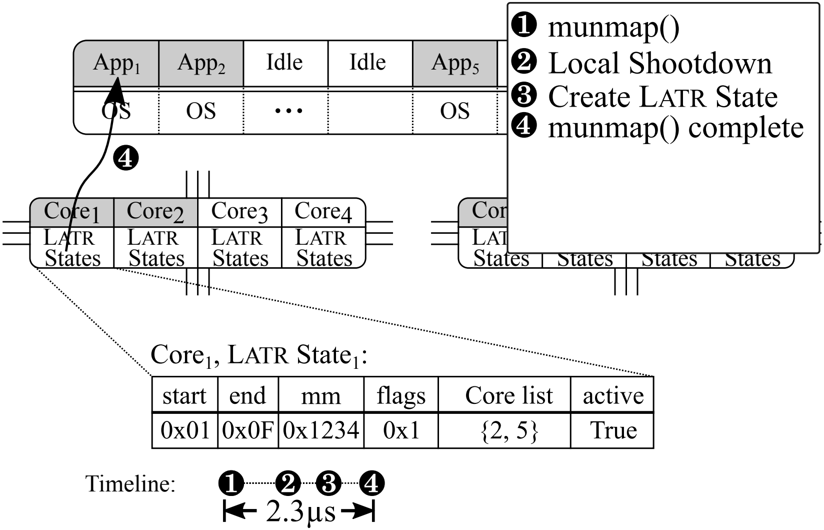
- Set up LATR state (for cores 2 and 5), local shootdown:



start	end	mm	flags	Core list	active
0x01	0x0F	0x1234	0x1	{2, 5}	True

LATR: Example

- Return control on core 1. Time taken: 2.3 μs, 70% reduction:

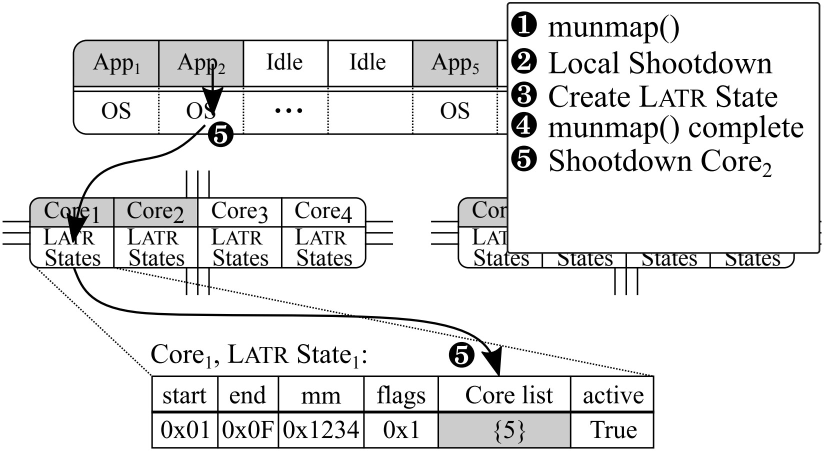


Core$_1$, LATR State$_1$:

start	end	mm	flags	Core list	active
0x01	0x0F	0x1234	0x1	{2, 5}	True

LATR: Example

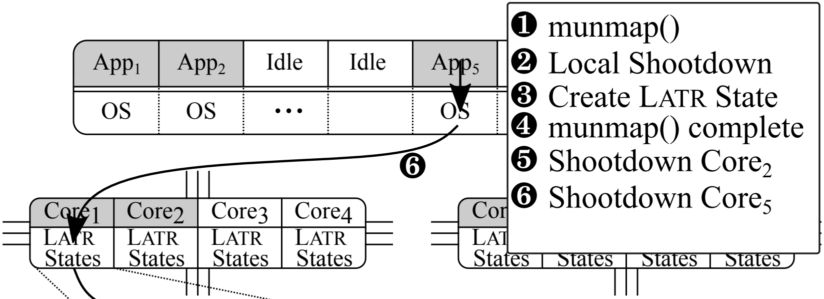
- Scheduler tick on core 2, local shootdown, reset state:



start	end	mm	flags	Core list	active
0x01	0x0F	0x1234	0x1	{5}	True

LATR: Example

- Scheduler tick on core 5, local shootdown, reset state:



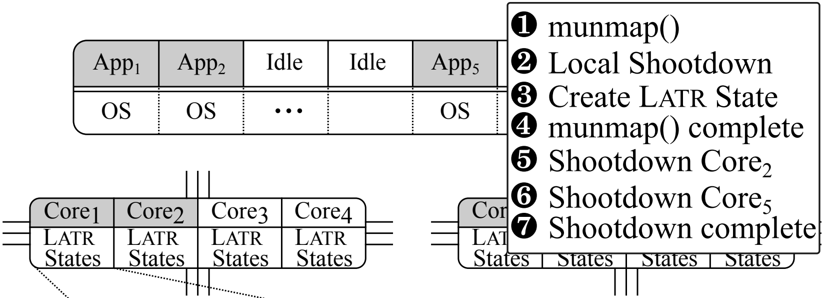
❶ munmap()
❷ Local Shootdown
❸ Create LATR State
❹ munmap() complete
❺ Shootdown $Core_2$
❻ Shootdown $Core_5$

$Core_1$, LATR $State_1$:

start	end	mm	flags	Core list	active
0x01	0x0F	0x1234	0x1	{}	False

LATR: Example

- Shootdown complete, LATR entry can be reused:



Core$_1$, LATR State$_1$:

start	end	mm	flags	Core list	active
0x01	0x0F	0x1234	0x1	{}	False

Lazy TLB shootdown: Correctness

- Same physical memory or virtual memory is reused
    - Leads to memory corruption
- $\Rightarrow$ Avoid same physical/virtual page reuse
    - Upper bound for TLB shootdown with LATR is 1ms
    - OS physical/virtual memory reclamation delayed by two scheduler ticks (2ms)
    - Memory overhead is bounded by 21 MB

Lazy TLB shootdown: Incorrect accesses

- Memory accesses before LATR shootdown:
  - Consequence of incorrect application: Use After Free
  - Before LATR shootdown, access (reads and writes) allowed
  - Exists in the current OS implementation
  - After LATR shootdown, access results in segmentation fault

Scope of LATR

- ABI change for *free* operations
- Support for operations limited to few, frequently used operations:

Classification	Operations	Lazy operation possible
Free	`munmap()`: unmap address range `madvise()`: free memory range	✓ ✓
Migration	AutoNUMA page migration ($\Rightarrow$ See paper) Page swap: swap page to disk	✓ ✓
Permission	`mprotect()`: change page permission	-
Ownership	CoW: Copy on Write	-
Remap	`mremap()`: change physical address	-

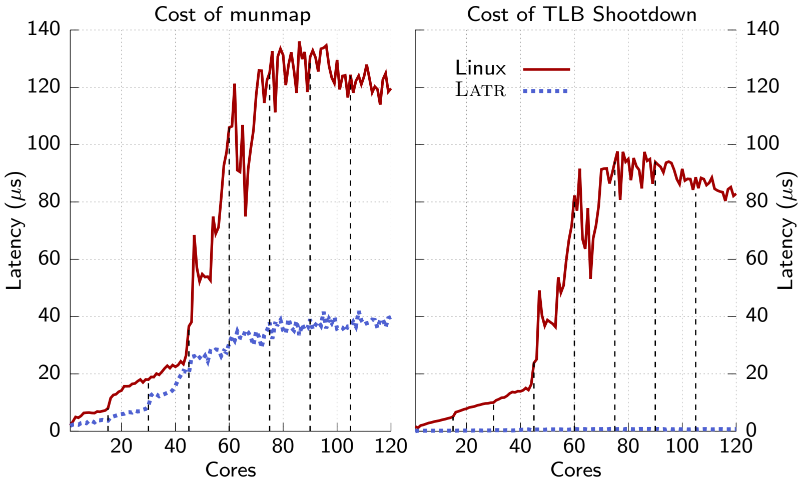
Table of contents

Evaluation: Questions

- LATR prototype developed for Linux 4.10
- Evaluation questions
  - What are LATR's benefits with microbenchmarks?
  - What are LATR's benefits with real-world applications exhibiting many TLB shootdowns?
  - What is the cost for LATR?

Microbenchmark on eight sockets

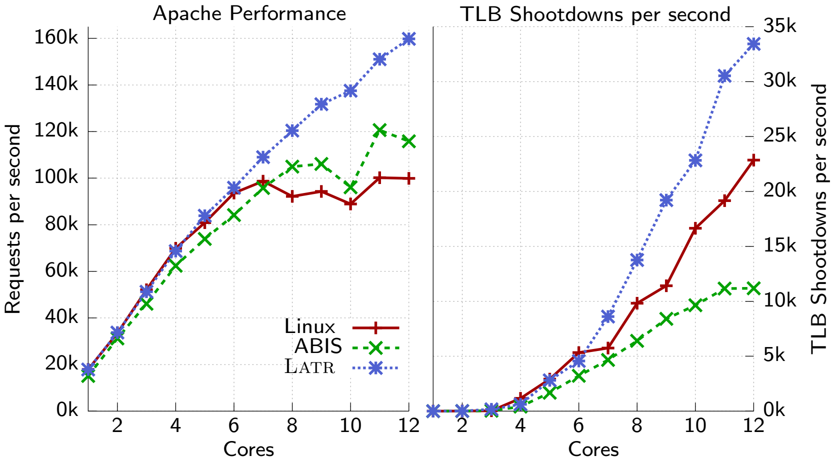
- Linux and LATR calling `munmap()` with one page on 120 cores:



⇒ **Up to 66.7% reduction for `munmap()`**

Serving files with Apache

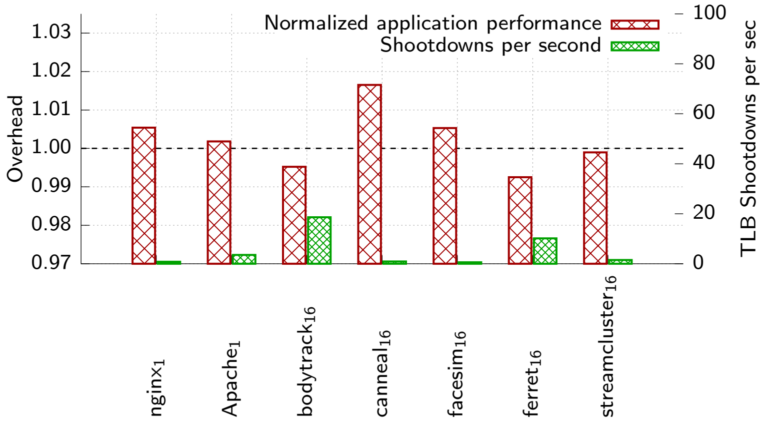
- Linux, ABIS [ATC17], and LATR on 2 sockets:



⇒ **Up to 59.9% more $\frac{\textbf{requests}}{\textbf{second}}$ than Linux, 37.9% higher than ABIS.**

Cost of LATR

- Memory overhead is bounded by 21 MB
- Performance overheads for applications with few TLB shootdowns:



⇒ LATR shows small performance overheads of up to 1.7% due to added operations during scheduling.

Future work

Further applications of LATR in:

- Disaggregated data centers
- Heterogeneous memory
- Applicability to PCID/ASID-based approaches
- Impact on new features such as KPTI, ...?

LATR: Takeaways

- The **synchronous** TLB shootdown is expensive
- We propose a software-based **asynchronous** shootdown mechanism
- Significant improvement in application performance with LATR
    - **70%** reduction for `munmap()`, for 16-core and 120-core machines
    - Improves Apache's throughput by **60%**
- Asynchronous mechanism applicable to other services:
    - AutoNUMA (see our paper)

LATR: Takeaways

- The **synchronous** TLB shootdown is expensive
- We propose a software-based **asynchronous** shootdown mechanism
- Significant improvement in application performance with LATR
  - **70%** reduction for `munmap()`, for 16-core and 120-core machines
  - Improves Apache's throughput by **60%**
- Asynchronous mechanism applicable to other services:
  - AutoNUMA (see our paper)

Thanks!