Optimizing the TLB Shootdown Algorithm with Page Access Tracking

Nadav Amit – VMware Research Group
Translation Lookaside Buffer (TLB)

- TLB = cache for virtual to physical address translations
TLB Coherency

- Hardware does not maintain TLBs coherent
- The problem is left for software (OS)
Local TLB Flushes and Remote TLB Shootdowns

- PTEs
- PTE change
- local flush
- local flush
- IPI
- Done
- TLB shootdown
When do TLB Flushes Occur?

- **Application initiated**
  - munmap()
  - Copy-on-write
  - msync()
  - mprotect()
  - madvise()
  - migrate_pages()

- **OS initiated**
  - NUMA migrations
  - Memory compaction
  - Memory deduplication
  - Memory reclamation
  - Memory balloon
  - Background dirty-pages flush

- Faster storage → TLB overheads are more apparent
Existing Solutions

- **Hardware** [Teller’90, Villavieja’11, Li’13]

- **Software (commodity OSes)**
  - Batching [Uhlig’05]
  - Limit flushes to cores that use the address-space
  - Trade-off between full and individual PTE flushes

- **Software (academic)**
  - Explicit software control [Boyd-Wickizer’10, Tene’11]
  - Replicated paging hierarchy [Clements’13, Gerofi’13]
Replicated Paging Hierarchy
[Clements’13, Gerofi’13]

- Page-fault on each CPU that accesses a PTE
- Memory overheads
- Runtime overheads: managing multiple tables
Insight: Use PTE Access-Bit

PTE: page frame number A permissions

• Set by hardware, cleared by software

• Used for OS memory reclamation decisions
  – Set when a page is accessed
  – “These flags are provided ... to manage the transfer of pages ... into and out of physical memory.” (Intel SDM)

• Insight: can be used for TLB invalidation decisions
  – Set when a PTE is cached
  – “Whenever the processor uses a PTE as part of address translation, it sets the accessed flag...” (Intel SDM)
Our System

• Flush decisions based on PTE access-bit

• Software solution (x86)
  – Exploiting the full potential requires simple hardware changes

• Prevent common unnecessary TLB shootdowns
  – Long-lived idle mappings
  – Short-lived private mappings

• Some false positives
  – Unnecessary flushes
Long-Lived Idle Mappings

CLOCK Algorithm
[Carr and Henessy ‘81]

- advance clock pointer
- test and clear A-bit
- PTE.A
  - Set
  - Clear
- dirty bit
  - Set
  - Clear
- TLB flush (up to 9us)
- replace page
- schedule page for cleaning

TLB flush (up to 9us)
Avoiding Flush of Long Lived Idle PTEs

- Test and clear A-bit
  - Set
  - Advance clock pointer
- All cores perform full TLB flush
- Test and clear A-bit
  - Clear
  - Advance clock pointer

PTE not cached
TLB Version Tracking (1)

- **PTE**
  - A-bit
  - 0
  - Clear A-bit

- **SPTE**
  - Version #
  - 5

- **Address Space**
  - Version #
  - 5
  - CPU Bitmap
  - 1 0 0 0

- Full TLB flush (CPU1)
- Clear bit
### TLB Version Tracking (2)

<table>
<thead>
<tr>
<th>PTE</th>
<th>SPTE</th>
</tr>
</thead>
<tbody>
<tr>
<td>A-bit</td>
<td>version #</td>
</tr>
<tr>
<td>0</td>
<td>5</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Address Space</th>
</tr>
</thead>
<tbody>
<tr>
<td>version #</td>
</tr>
<tr>
<td>6</td>
</tr>
</tbody>
</table>

- **Version**: `
  version CPU bitmap
  1 1 1 1`
- **CPU bitmap**
- **Full TLB flush (CPU3)**
- **Reset**
TLB Version Tracking (3)

PTE | SPTE
---|---
A-bit | version #
0 | 5

PTE unmap and flush:

If \( PTE.A == 0 \) and \( SPTE\text{.ver} + 1 < AS\text{.ver} \)

Then avoid TLB flush

Address Space

version #
7

version
CPU bitmap
1 0 1 1
Short Lived Private Mappings

mmap() -> read access -> mumap()

- page-fault
  - PTE set
- PTE cleared
  - TLB flush

usually same core

insert PTE to TLB with PTE.A=0

PTE.A==0?

local flush
Evaluations

• Prototype based on Linux 4.5

• Baseline configured to avoid shootdown cost
  – Linux version that uses TLB flushes batching
  – Using efficient multicast IPI delivery

• 48-cores, 2-socket server

• Our system denoted as ABIS: Access-Based Invalidation System
Apache TLB Shootdowns
(Short-Lived Private Mappings)
Apache Performance

![Graph showing Apache performance with baseline, ABIS, and speedup lines.](chart.png)

- **Requests/sec [thousands]**
- **Speedup**
- **Cores [#]**

Legend:
- **Baseline**
- **ABIS**
- **Speedup**
PBZIP2 – TLB shootdowns
(Long-Lived Idle Mappings)
PBZIP2 Performance

![Graph showing PBZIP2 performance with runtime in seconds on the y-axis and threads on the x-axis. The graph compares baseline and ABIS speedup.]
Microbenchmarks: VMScale
Conclusions

• Access-bit tracking can often prevent most TLB shootdowns:
  – Long-lived idle PTEs
  – Short-lived private PTEs

• Exploit memory coherency to check if TLB is cached

• CPUs should allow more control over the TLB
  – Insertion of PTEs directly to the TLB