Virtual Memory Management: TLB Prefetching & Page Walk

Yuxin Bai, Yanwei Song

CSC456 Seminar

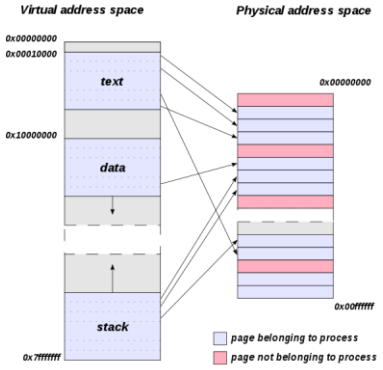
Nov 3, 2011

What's Virtual Memory Management

- Illusion of having a large amount of memory
- Protection from other programs
- Data sharing with other programs

Problems to be handled

- Program uses virtual (logical) address
- Memory uses physical address to store the actual data
- Address Translations are handled by MMU in between
  - Better to Hit the TLB
  - If Miss the TLB, it's better to walk the Page Table faster



Memory Management Unit (MMU)

- Hardware for address translation
  - Translation Look-aside Buffer (TLB)
    - Fully/Highly associative
    - Typically accessed every cycle!! (normally in parallel with L1 cache, virtually indexed)
    - Software/Hardware Management
  - Page Walker

Table 1. Comparison of architectural support for virtual memory in six commercial MMUs.

Item	MIPS	Alpha	PowerPC	PA-RISC	UltraSPARC	IA-32
Address space protection	ASIDs	ASIDs	Segmentation	Multiple ASIDs	ASIDs	Segmentation
Shared memory	GLOBAL bit in TLB entry	GLOBAL bit in TLB entry	Segmentation	Multiple ASIDs; segmentation	Indirect specification of ASIDs	Segmentation
Large address spaces	64-bit addressing	64-bit addressing	52-/80-bit segmented addressing	96-bit segmented addressing	64-bit addressing	None
Fine-grained protection	In TLB entry	In TLB entry	In TLB entry	In TLB entry	In TLB entry	In TLB entry; per segment
Page table support	Software-managed TLB	Software-managed TLB	Hardware-managed TLB; inverted page table	Software-managed TLB	Software-managed TLB	Hardware-managed TLB/hierarchical page table
Superpages	Variable page size set in TLB entry: 4 Kbyte to 16 Mbyte, by 4	Groupings of 8, 64, 512 pages (set in TLB entry)	Block address translation: 128 Kbytes to 256 Mbytes, by 2	Variable page size set in TLB entry:4 Kbytes to 64 Mbytes, by 4	Variable page size set in TLB entry: 8, 64, 512 Kbytes, and 4 Mbytes	Segmentation/ variable page size set in TLB entry: 4 Kbytes or 4 Mbytes

[Jacob.IEEEMicro'98]

TLB Miss Handling Comparison

	Software Management	Hardware Management
Miss handler	10-100 instructions long	Finite-state Machine
Instruction cache pollution	Yes	No
Data cache pollution	Yes	Yes
Rigid page organization	No	Yes
Pipeline Flushing	Yes	No

The performance differences are not large enough to prefer one over another, standardization on support of virtual memory system is suggested. [Jacob.asplos98]

TLB Miss Handling Cost

- TLB miss handling at 5-10% of system runtime, up to 40% runtime [Jacob.aslpos98]
- DTLB miss handling can amount to 10% of the runtime of SPEC CPU2000 workloads [Kandiraju.sigmetric02]
- As the physical and virtual addresses grow in size, the depth of page table levels increase with generations, thus the number of memory accesses increases for a single address translation, which increases TLB miss penalty
  - e.g. 2 levels in Intel 80386, 3 levels in Pentium Pro, 4 levels in AMD Optern and Intel x86-64

TLB optimizations

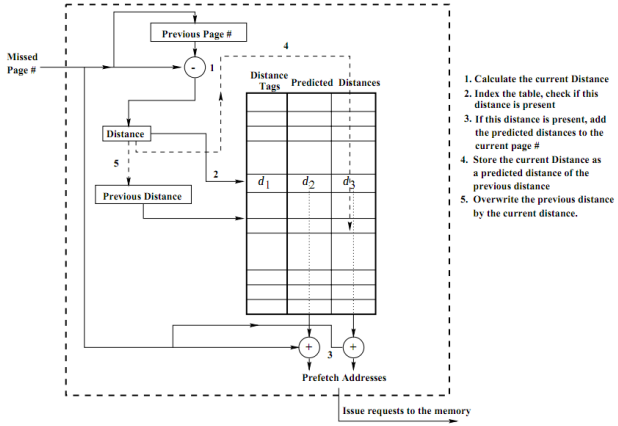
TLB hardware conventional optimizations

- TLB size, associativity, and multi-level hierarchy [Chen.isca92]
- Super Page [Talluri.95]
- TLB prefetching

TLB prefetching

- Software prefetches entries on Inter-Process Communication path[Kavita.sosp94]
  - For communicating processes, prefetch entries mapping IPC data structures/message buffers, stack and code segments
- Distance Prefetching [Kandiraju.isca02]
  - 1,2,4,5,7,8: page # of TLB miss
    (1,2) (2,1) : distance pairs would be tracked
    (distance, predict distance)

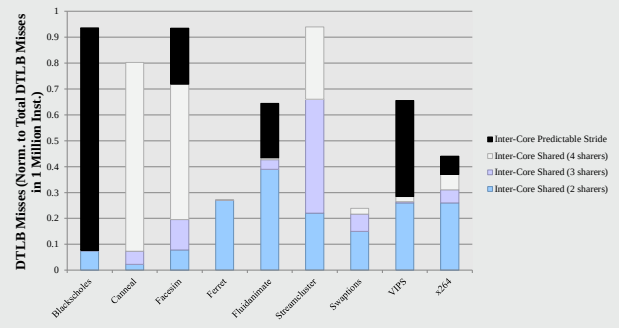
Distance Based Prefetching



Challenges and Opportunity in CMP

- Challenges
  - Novel parallel workloads stress TLBs heavily [Bhattacharjee.pact09]
  - TLB consistency among multiprocessors (TLB shoot-down needed)
- Opportunity
  - Parallel workload also exhibit commonality in TLB misses across cores

Categorized common patterns



Goal: Use commonality in miss patterns to prefetch TLB entries to cores based on the behavior of other cores

Courtesy of Abhishek Bhattacharjee's ppt

Inter-Core Cooperative TLB Prefetchers for Chip Multiprocessors

- Explore two types of inter-core cooperative TLB prefetchers individually and then combine them (specifically for DTLBs)
- Prefetcher 1: Leader-Follower TLB Prefetching
  - Targeted at Inter-core Shared TLB Misses
- Prefetcher 2: Distance-Based Cross-Core Prefetching
  - Targeted at Inter-core Shared and Inter-core Stride TLB Misses

Courtesy of Abhishek Bhattacharjee's ppt

Leader-Follower TLB Prefetching

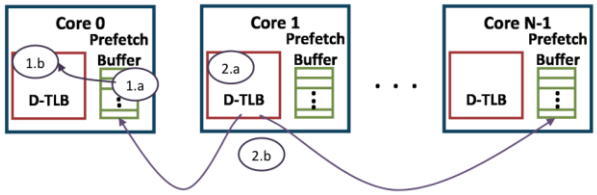
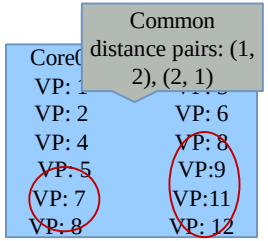


Figure 2. The baseline Leader-Follower algorithm prefetches a TLB miss translation seen on one core (the leader) into the other cores (the followers) to eliminate inter-core shared TLB misses.

Courtesy of Abhishek Bhattacharjee's ppt

Distance-Based Cross-Core Prefetching

- Targeted at eliminating inter-core stride TLB misses
- The idea: use distances between subsequent TLB miss virtual pages from single core to prefetch entries



This approach can capture various stride patterns among cores effectively (and also capture within-core TLB miss patterns)

Courtesy of Abhishek Bhattacharjee's ppt

Distance-Based Prefetching HW

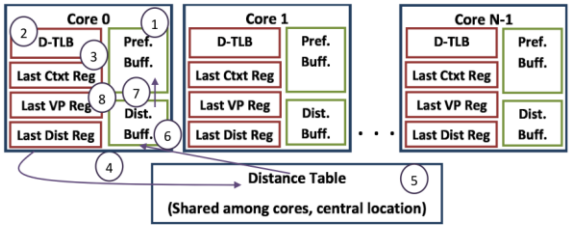
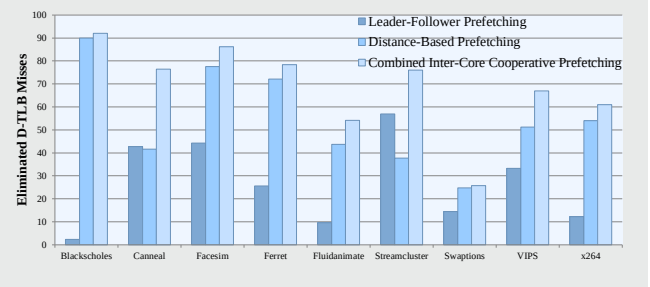


Figure 5. Distance-based Cross-Core prefetching uses a central, shared Distance Table to store distance pairs and initiates prefetches based on these patterns whenever a TLB miss occurs on one of the cores (for both PB hits and misses). Note that the prefetches on a core may be initiated by a distance-pair initially seen on a different core.

Courtesy of Abhishek Bhattacharjee's ppt

Combining the Two Prefetching Schemes

- Keep infinite size prefetch buffer and see how combining leader-follower and distance-prefetching (512-entry) does



Courtesy of Abhishek Bhattacharjee's ppt

Conclusion

- The TLB miss cost is high
- In CMPs, TLB consistency needs be maintained (We didn't go into depth)
- Common TLB miss patterns in parallel workloads can be exploited
- Many approches (prefetching, sharing) are proposed

Page Table Walk Accelerations

x86 Address Translation

- Since 80386, all Intel processors use a radix tree to record the hierarchical mapping from virtual to physical addresses. (64-bit virtual address, 4-level page table)
- Four levels Page Table
  - L4: Page Map Level 4 (PML4)
  - L3: Page Directory Pointer (PDP)
  - L2: Page Directory (PD)
  - L1: Page Table (PT)

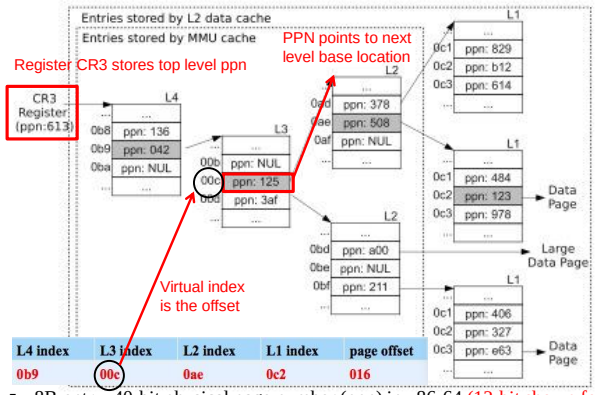
An example: Walk the Page Table

- Virtual address (x86-64): <0x0000, 5c83, 15cc, 2016>
  - 12-bit offset for 4KB page size, four 9-bit indices to select entries from the 4 levels of the page table, 16-bit sign-extended

63:48	47:39	38:30	29:21	20:12	11:0
se	L4 idx	L3 idx	L2 idx	L1 idx	*page offset*

- Re-Write Virtual Address in the format as:
  <L4 index, L3 index, L2 index, L1 index, page offset>

L4 index [47:39]	L3 index [38:30]	L2 index [29:21]	L1 index [20:12]	page offset [11:0]
0b9	00c	0ae	0c2	016



- 8B-entry, 40-bit physical page number (ppn) in x86-64 (12-bit shown for simplicity)
- It requires 4 memory references to walk the page table to translate an address

The Idea of MMU caches

- A single address translation needs multiple memory accesses, but the accesses to upper level page table entries can have great temporal locality
  - e.g. two consecutive accesses have the same 3 upper level entries, only the last level is different
- Caching MMU upper level page table entries could possibly reduce the TLB miss penalty (page walk overhead) by decreasing the number of memory accesses

Caching Page Walks: the design space

- 1. Page Table Caches
  - 1.1 Unified Page Table Cache (UPTC)
  - 1.2 Split Page Table Cache (SPTC)
- 2. Translation Caches
  - 2.1 Split Translation Cache (STC)
  - 2.2 Unified Translation Cache (UTC)
- 3. Translation-Path Caches (TPC)

1.1 Unified Page Table Cache

- UPTC (e.g. AMD's Page Walk Cache)
- Entries from different levels of the page table are mixed, all indexed by their physical address

Base Location	Index	Next Page
125	0ae	508
042	00c	125
613	0b9	042
...	...	...

- e.g. To translate next Virtual address (0b9, 00c, 0ae, 0c3, 103)
  - Search 0b9 in the cache, locate next page table number is 042
  - Search 042, 00c, locate next page table number is 125
  - …
  - Search 508, 0c3, not match, now go to the page table (level L1)
- Only one memory access, 3 upper level entries are hit in the cache

1.2 Split Page Table Cache

- SPTC

	Base Location	Index	Next Page
L2 entries	125 ...	0ae ...	508 ...
L3 entries	042 ...	00c ...	125 ...
L4 entries	613 ...	0b9 ...	042 ...

- Each entry holds the same tag and data as in UPTC
- Separate into several caches for different levels, so that entries from different levels do not compete for common slots

2.1 Split Translation Cache

- STC (Intel Paging-Structure Caches, in modern Intel x86-64 processors)
  - Tagged by virtual indices, storing the partial translation (next base ppn)
  - MMU selects the longest prefix match to skip the most levels
  - Separate caches for different level page table pages

	L4 index	L3 index	L2 index	Next Page
L2 entries	0b9 ...	00c ...	0ae ...	508 ...
L3 entries	0b9 ...	00c ...		125 ...
L4 entries	0b9 ...			042 ...

- e.g. To translate next Virtual address (0b9, 00c, 0dd, 0c3, 929)
  - Search (0b9), (0b9, 00c), (0b9, 00c, 00d) … in any order or in parallel
  - (0b9, 00c) is found as the longest prefix matched
- Only 2 memory accesses are needed in this case

2.2 Unified Translation Cache

- UTC
  - Elements from all levels are mixed in a single cache

L4 index	L3 index	L2 index	Next Page
0b9	00c	0ae	508
0b9	00c	xx	125
0b9	xx	xx	042
...	...	...	...

- An "xx" means "don't care"
- Select a longest prefix if there are multiple matches

3. Translation-Path Caches

- TPC

L4 index	L3 index	L2 index	L3	L2	L1
0b9	00c	0ae	042	125	508
...	...	...	...	...	...

- A single entry represent an entire path, tagged by 3 9-bit indices, store 3 ppns
- e.g. After the translation of (0b9, 00c, 0ae, 0c2, 016), all data from that walk is stored in one entry
  - If next VA is (0b9, 00c, 0ae, 0c3, 929), match (0b9, 00c, 0ae), return L1 ppn 508
  - If next VA is (0b9, 00c, 0de, 0f, 829), match (0b9, 00c), return L2 ppn 125

Evaluating each design possibility

- Which part is used for tag?
  - Virtual (shorter, search in any order or parallel)
  - Physical (40-bit, sequential search)
- What kind of partition strategy?
  - Unified (sparsely use Virtual Address Space)
  - Split on levels (densely)
- Complexity?
  - Tag array (fully-associative CAM, power hungry)
  - Data array (RAM)

Conclusion

- MMU caches have become critical components in x86 processors (hierarchical page table structure)
- Translation Cache is better than others
  - Virtual tagged (smaller, lower power), address partially translated (faster)
- Unified Translation cache designed in this paper is better than Intel's split translation cache and AMD's unified page table cache
  - Adapt to varying workloads

Summary

- TLB miss handling cost is a significant component in the total runtime
- To improve TLB miss rate, one of the approaches, prefetching is covered
- To accelerate TLB page walk, page table cache is introduced

Reference

- A. Bhattacharjee and M. Martonosi, Inter-Core Cooperative TLB Prefetchers for Chip Multiprocessors, ASPLOS 2010
- B. Jacob and T.Mudge, A look at several memory management units, TLB-refill mechanisms, and page table organizations
- B. Jacob and T.Mudge, Virtual Memory in Contemporary Microprocessors
- Thomas W. Barr, Alan L. Cox, Scott Rixner, Translation Caching: Skip, Don't Walk (the Page Table), ISCA 2010

UNIVERSITY of ROCHESTER