ECE 571 – Advanced Microprocessor-Based Design Lecture 16

Vince Weaver

http://www.eece.maine.edu/∼vweaver

vincent.weaver@maine.edu

4 November 2014

Announcements

- HW4 due Friday
- Also project topic selection due Friday

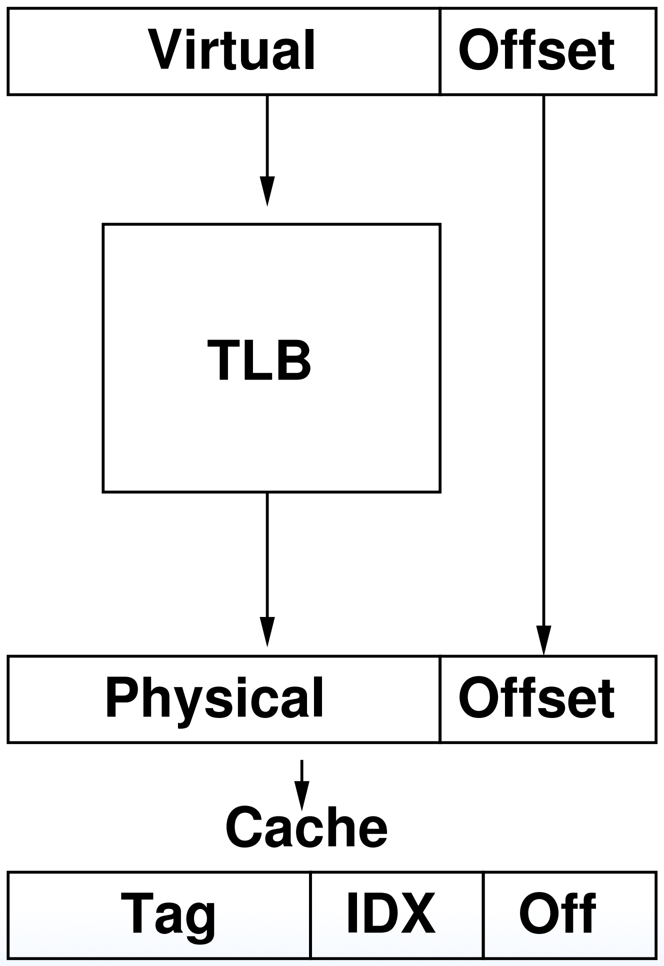
Virtual Memory – Cache Concerns

Cache Issues

- Page table Entries are cached too
- What happens if more memory can fit in the cache than can be covered by the TLB?
- If you have 128 TLB entries * 4kB you can cover 512kB
- If your cache is larger (say 1MB) then a simple walk through the cache will run out of TLB entries, so page lookups will happen (bringing page table data into cache) and so you do not get maximal usefulness from the cache

- This has happened in various chips over the years

Physical Caches

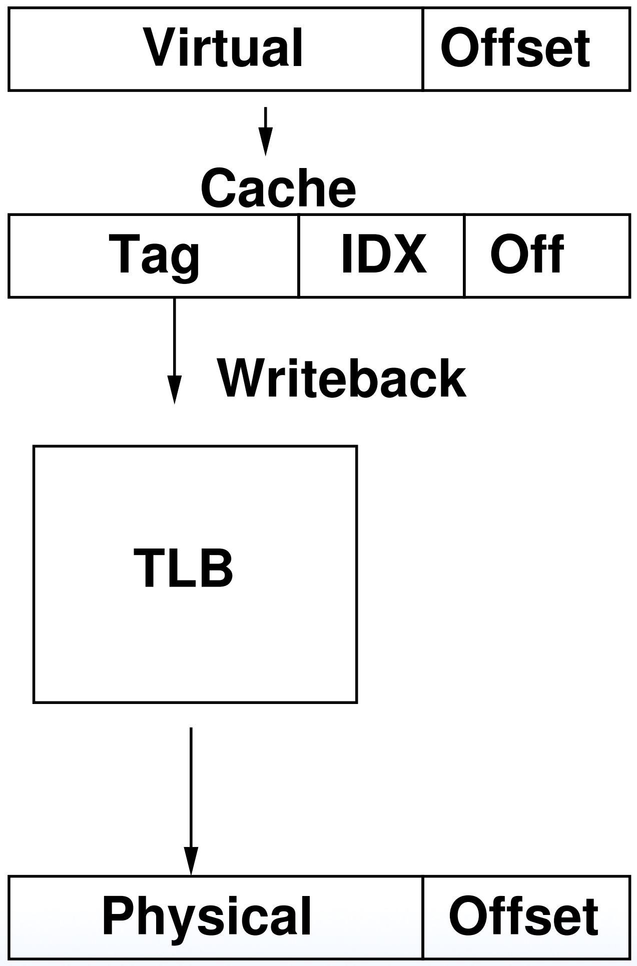


Physical Caches

- Location in cache based on physical address
- Can be slower, as need TLB lookup for each cache access
- No need to flush cache on context switch (or ever, really)
- No need to do TLB lookup on writeback
- If properly sized, the index bits are the same for virt and physical. In this case no need to do TLB lookup on cache hit.

- If not sized, the extra index bits need to be stored in the cache so they can be passed along with the tag when doing a lookup

Virtual Caches



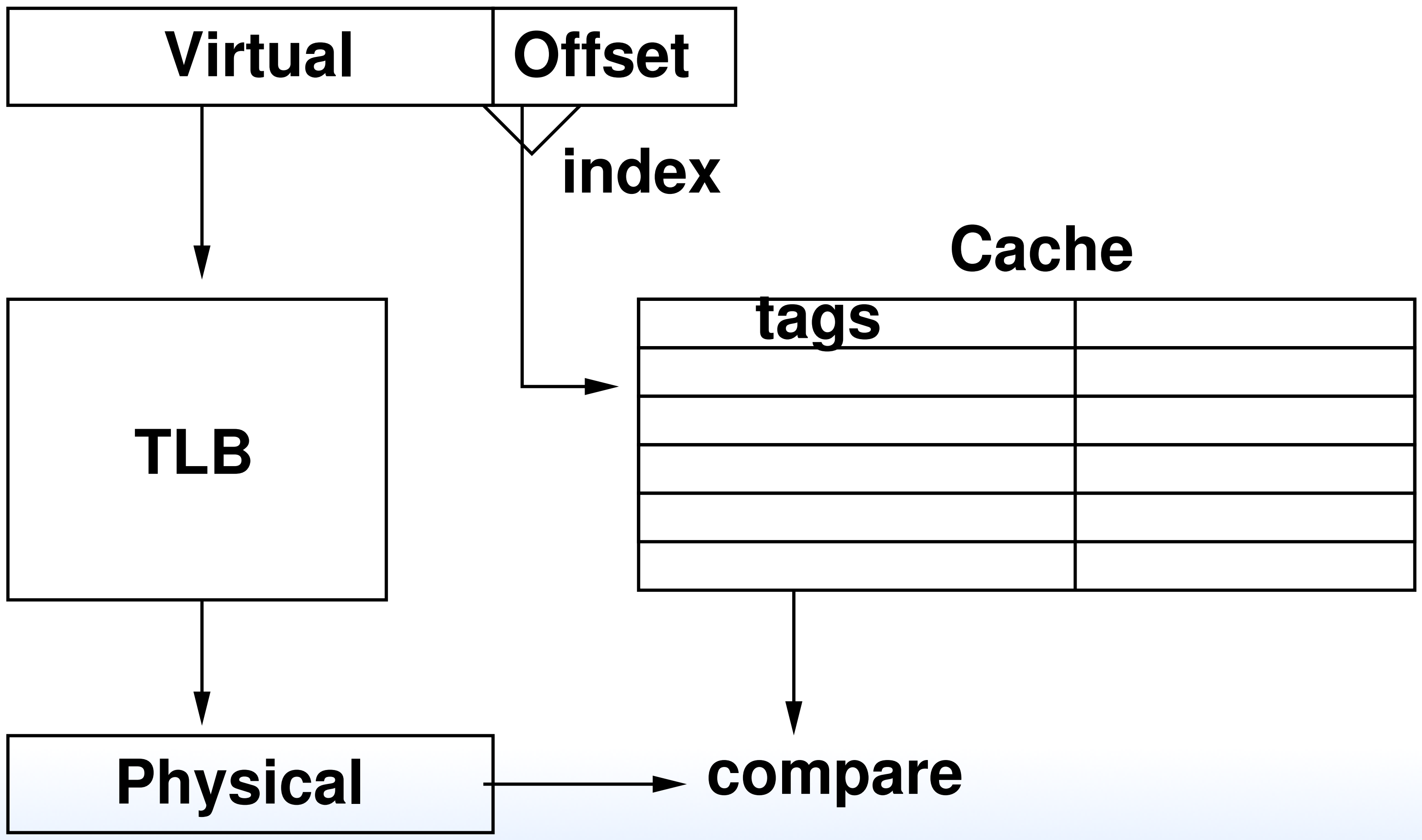
Virtual Caches

- Location in cache based on virtual address
- Faster, as no need to do TLB lookup before access
- Will have to use TLB on miss (for fill) or when writing back dirty addresses
- Cache might have extra bits to indicate permissions so TLB doesn't have to be checked on write
- Can have aliasing issues when processes use same virtual

addresses.
Flush cache on context switch?

- How to avoid flushing? Have a process-id. Can also implement sharing this way, by both processes mapping to same virt address.
- Having kernel addresses high also avoids aliasing
- Operating system has to do more work

VIPT



Cache lookup and TLB lookup in parallel. Cache size $+$ associativity must be less than page size.

Combinations

- PIPT – older systems. Slow, as must be translated (go through TLB) for every cache access (don't know index or tag until after lookup)
- VIVT – fast. Do not need to consult TLB to find data in cache.
- VIPT – ARM L1/L2. Faster, cache line can be looked up in parallel with TLB. Needs more tag bits.
- PIVT – theoretically possible, but useless. As slow as

PIPT but aliasing like VIVT.

Large Pages

- Another way to avoid problems with 64-bit address space
- Larger page size (64kB? 1MB? 2MB? 2GB?)
- Less granularity. Potentially waste space
- Fewer TLB entries needed to map large data structures
- Compromise: multiple page sizes.
  Complicate O/S and hardware. OS have to find free blocks of contiguous memory when allocating large page.

- Transparent usage? Transparent Huge Pages? Alternative to making people using special interfaces to allocate.

Haswell Virtual Memory

- L1 (4-way associative)
  - 64 4kB
  - 32 2MB
  - 4 1GB
- L2 (1024 entry 8-way associative, combined 4kB and 2M)
- DCache – 32kB/8-way so VIPT possible

Cortex A9 MMU

- Virtual Memory System Architecture version 7 (VMSAv7)
- page table entries that support 4KB, 64KB, 1MB, and 16MB
- global and address space ID (no more TLB flush on context switch)
- instruction micro-TLB (32 or 64 fully associative)

- data micro-TLB (32 fully associative)
- Unified main TLB, 2-way, 2x64 (128 total) on pandaboard
- 4 lockable entries (why want to do that?)
- Supports hardware page table walks

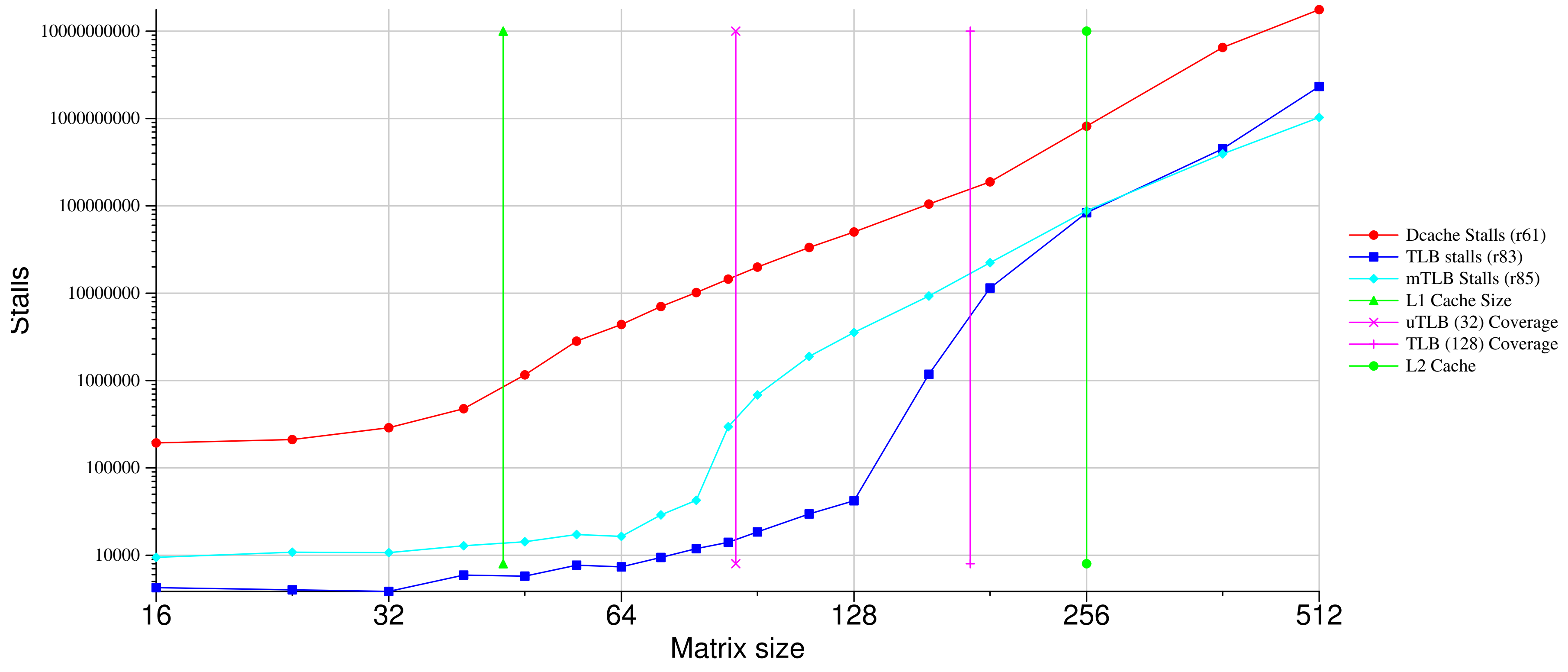
Cortex A9 MMU

- Virtual Memory System Architecture version 7 (VMSAv7)
- Addresses can be 40bits virt / 32 physical
- First check FCSE – linear translation of bottom 32MB to arbitrary block in physical memory (optional with VMSAv7)

Cortex A9 TLB

- micro-TLB. 1 cycle access. needs to be flushed if ASID changes
- fully-associative lockable 4 elements plus 2-way larger. varying cycles access

Cortex A9 TLB Measurement



Having Larger Physical than Virtual Address Space

- 32-bit processors cannot address more than 4GB
  x86 hit this problem a while ago, ARM just now
- Real solution is to move to 64-bit
- As a hack, can include extra bits in page tables, address more memory (though still limited to 4GB per-process)
- Linus Torvalds hates this.

- Hit an upper limit around 16-32GB because entire low 4GB of kernel addressable memory fills with page tables

TLB Energy

Simple ideas

- TLB is similar to a cache, so use same optimization techniques (drowsy, etc)

TLB Optimization – Assume in Same Page

- (Kadayif, Sivasubramaniam, Kandemir, Kandiraju, Chen. TODAES 2005).
  Don't access TLB if not necessary. Compare to last access (assume stay in same page) Circuit improvements

- (Kadayif,Sivasubramaniam, Kandemir, Kandiraju, Chen. Micro 2002)
  Cache page value.

TLB Optimization – Use Virtual Caches

- (Ekman and Stenström, ISLPED 2002) Use virt address cache. Less TLB energy, more snoop energy. TLB keeps track of shared pages.

TLB Optimization – Reconfiguring

- (Basu, Hill, Swift. ISCA 2012) Have the OS select if memory region physical or virtual cached.
- (Delaluz, Kandemir, Sivasubramaniam, Irwin, Vijaykrishnan. ICCD 2013) Reducing dTLB Energy Through Dynamic Resizing.
  Size TLB as needed, shutting off banks. Easier if fully-associative.

TLB Optimization – Memory Placement

- (Jeyapaul, Marathe, Shrivastava, VLSI'09) Try to keep as much in one page as possible via compiler.

- (Lee, Ballapuram. ISLPED'03) Split memory regions by region (text/data/heap). Better TLB performance, better energy.