Cache Hierarchy

Cache hierarchy requirements

- different caches with different sizes & access times & purposes
  - **level 1 cache -- goal: fast access,**
    so minimize hit time (the common case)
    - small, so can access it in one CPU cycle
      (also chip real estate constraints)
    - virtually-addressed, so cache accesses can be fast without constraints on cache size
    - direct mapped or set associative?
      - direct mapped: faster access
      - set associative: better hit ratio
    - separate caches for instructions & data
      - each is smaller than a unified cache, so the access time is lower
      - configured differently:
        instruction cache has larger blocks
        instruction cache has no memory update hardware

Cache Hierarchy

- different caches with different sizes & access times & purposes
  - **level 2 -- goal: keep traffic off the system bus**
    - big cache, so it will have a high hit ratio
    - physically-addressed:
      - plenty of time to do address translation
      - no flushing on a context switch
      - snooping *(later)*
    - direct mapped or set associative?
      - same trade-off, same lack of consensus on the design
      - big direct-mapped caches have almost the same hit ratio as big set associative caches
      - but the cache access time is much greater than the MUX time
    - unified, because its hit ratio is higher than that of two separate caches (I&D) half the size
    - write-back
      - only use bus for dirty blocks on a block replacement

How does a cache hierarchy improve cache performance?

Measuring Cache Hierarchy Performance

Effective Access Time:

$$\text{hit time}_{L1} + \text{miss ratio}_{L1} \bullet \text{misspenalty}_{L1}$$

$$\text{hit time}_{L2} + \text{miss ratio}_{L2} \bullet \text{misspenalty}_{L2}$$

Measuring Cache Hierarchy Performance

Local Miss Ratio:

$$\frac{\#\text{misses}}{\#\text{accesses}} \text{ for that cache!}$$

- # accesses for the L1 cache: the number of references
- # accesses for the L2 cache: the number of misses in the L1 cache

Example: 1000 references
40 L1 misses
10 L2 misses

local MR (L1):

local MR (L2):

Measuring Cache Hierarchy Performance

Global Miss Rate:

$$\textbf{globalMR} = \frac{\text{\# misses in cache}}{\text{\# references generated by CPU}}$$

Example: 1000 references
40 L1 misses
10 L2 misses

global MR (L1):

global MR (L2):

global MR:

Handling a Cache Miss

(1) Send the address (PC-4 or effective address) to the next level of the hierarchy

(2) SIgnal a read operation

(3) Wait for the data to arrive

(4) Update the cache entry with data*, rewrite the tag, turn the valid bit on

(5) Reaccess the cache. This time it will hit.

* There are variations:

- fill the cache block, then send the requested word to the CPU
- send the requested word to the CPU as soon as it arrives at the cache (**early restart**)
- requested word is sent from memory first; then the rest of the block follows (**requested word first**)

early restart and requested word first have a lower miss penalty, because they return execution control to the CPU earlier

Non-blocking Caches

Non-blocking cache (lockup-free cache)

- allows the CPU to continue executing instructions while a miss is handled
- some processors allow only 1 outstanding miss ("hit under miss")
- some processors allow multiple misses outstanding ("miss under miss")
- can be used with both in-order and out-of-order processors
  - **in-order processors** stall when an instruction that uses the load data is the next instruction to be executed (nonblocking loads)
  - **out-of-order processors** can execute instructions after the load consumer

How do non-blocking caches improve cache performance?

Non-blocking Cache Implementation

Miss status holding registers (MSHR)

- physical address of the block
- which word in the block
- destination register number
- mechanism to merge requests to the same block
- mechanism to insure accesses to the same location execute in program order

Sub-block Placement

Divide a block into sub-blocks

tag	V	data	V	data	V	data	V	data
tag	V	data	V	data	V	data	V	data
tag	V	data	V	data	V	data	V	data
tag	V	data	V	data	V	data	V	data

- **sub-block** = unit of transfer on a cache miss
- **valid bit**/sub-block

+ as much spatial locality as the whole block
    - but less implicit prefetching
+ the transfer time of a sub-block (good for read misses)
+ possibly decrease write time
+ fewer tags than if each sub-block were a block
- valid bit per subblock

Misses:

- block-level miss: tags didn't match
- sub-block-level miss: tags matched, valid bit was clear

How does sub-block placement improve cache performance?

Victim Caches

Victim cache

- small fully-associative cache
- contains the most recently replaced blocks of a direct-mapped cache
- check it on a cache miss
- alternative to 2-way set-associative cache
    - used with direct-mapped caches

Used on Alphas

How do victim caches improve cache performance?

Why do they work?

Pseudo-set Associative Caches

Pseudo-set associative cache

- access the cache as though it were direct mapped
- if miss, do a second access with the high-order index bit flipped
- prediction bit for which set to check

How does sub-block placement improve cache performance?

+ miss rate of 2-way set associative cache
+ close to access time of direct-mapped cache

Pipeline the Cache Access

Pipeline cache access

- 2 stages
  - cache access
  - ship data where it needs to go on the chip
- often used when the L1 cache access is 2 cycles

How does sub-block placement improve cache performance?

Prefetching

Prefetching

- fetch instructions and/or data before they are needed
- need a non-blocking cache

- instructions
  - fetch the next sequential instructions
- data
  - stride-based prefetching of arrays

- **stream buffers**
  - where prefetched instructions/data held
  - if requested block in the stream buffer, then cache access is cancelled & another prefetch is done instead

How does prefetching improve cache performance?

Review of Address Translation

Address translation:

- maps a virtual address to a physical address, using the page tables
- high-order bits translated; page offset bits the same for pages & page frames

Translation Lookaside Buffer (TLB):

- associative cache of most recently translated virtual-to-physical page mappings
  - 32-128-entry, fully associative to 4K-entry direct-mapped
  - 4-8 byte blocks
  - .5 -1 cycle hit time
  - low tens of cycles miss penalty
  - misses can be handled in software
    some are handled in hardware
- contents
  - physical page frame number
  - valid bit, dirty bit, reference bit
  - access information
  - process identifier
- tag is virtual page number or part of virtual page number
- works because of locality of reference within a page
- *much* faster than address translation using the page tables

Using a TLB

(1) Access the TLB using the virtual page number.

(2) If a **hit**,
concatenate the physical page number & the page offset bits, to form a physical address;
set the **reference bit**;
if writing, set the **dirty bit**.

(3) If a **miss**,
get the physical address from the page tables;
evict a TLB entry & update dirty/reference bits in the page tables;
update the TLB with the new mapping.

Virtual or Physical Addressing

Virtually-addressed caches:

- access with a virtual address (index & tag)
- do address translation on a cache miss

\+ faster for hits because no address translation

\- need to flush the cache on a context switch
    + process identification (PID) can avoid this

\- synonyms
    - **"the synonym problem"**
        - if 2 processes are sharing data, two (different) virtual addresses map to the same physical address
        - 2 copies of the same data could reside in the cache (these are the synonyms)
        - on a write, only one will be updated; so the other has old data
    - a solution: memory allocation restrictions
        - require processes to share segments, so all synonyms have the same low order bits
        - cache must be <= the segment size
        - all synonyms will then map to the same cache location
    - **If the cache is too small, how can you make it bigger?**

Virtual or Physical Addressing

Physically addressed caches

- access with a physical address (index & tag)
- do address translation on every access
- \- increase in hit time because must translate the virtual address before access the cache

    + increase in hit time can be avoided if address translation is done in parallel with the cache access
        - restrict cache size so that cache index bits are in the page offset
          (page offset bits are the same for virtual & physical addresses)
        - compare the physical tag from the cache to the physical address (page frame #) from the TLB
        - **If the cache is too small, how can you make it bigger?**
        - can increase cache size, but still use page offset bits for the index, by increasing associativity

+ no cache flushing on a context switch
+ no synonym problem

Machine Comparison

L1 on-chip instruction cache

Alpha 21264	Pentium Pro	UltraSparc 1
64KB 2-way 32B block	8KB 4-way 32B	16KB pseudo 2-way 32B block
2-cycle, pipelined access	1-cycle access	1-cycle access
virtual index, virtual tags TLB in parallel	virtual index, physical tags TLB in parallel	virtual index, virtual tags
miss under miss (8 outstanding misses w. D $)	nonblocking	?
prefetching with stream buffers		

Machine Comparison

L1 on-chip data cache

Alpha 21164	Pentium Pro	UltraSparc 1
64KB 2-way 64B block	8KB 2-way 32B block	16KB direct- mapped 32B block 16B subblock
2-cycle, pipelined access	1 cycle	1 cycle
virtual index physical tags TLB in parallel	virtual index physical tags TLB in parallel	virtual index physical tags TLB in parallel
write-back	write-back	write-through store compres-sion
miss under miss (8 outstanding misses w. I $)	nonblocking (hit under miss)	miss under miss (4)
dual-ported for reads	2 ports/4 banks	
8-entry victim $		

Machine Comparison

L2 cache

Alpha 21264	Pentium Pro	UltraSparc 1
1MB-16MB direct-mapped 64B blocks	512KB-4MB direct-mapped 8B block	256KB 4-way 32B block
	pipelined access	
unified	unified	
physical	physical	physical index physical tags
write-back	write-back	
?	?	nonblocking
		requested work first

Machine Comparison

TLBs

Alpha 21164	Pentium Pro
64 entries for both data & instructions	64 entries for data/32 for instructions
fully associative	4-way
can map 1, 8, 64, 512 8KB pages	
	miss handled in hardware