Introduction

Why memory subsystem design is important
• CPU speeds increase 25%-30% per year
• DRAM speeds increase 2%-11% per year

Memory Hierarchy

Levels of memory with different sizes & speeds
• close to the CPU: small, fast access
• close to memory: large, slow access

Memory hierarchies improve performance
1. **caches**: demand-driven storage
2. principal of **locality of reference**
   • **temporal**: a referenced word will be referenced again soon
   • **spatial**: words near a reference word will be referenced soon
3. speed/size trade-off in technology
   ⇒ **fast access for most references**

First Cache: IBM 360/85 in the late ‘60s
Cache Organization

Block:
- # bytes associated with 1 tag
- usually the # bytes transferred on a memory request

Set: the blocks that can be accessed with the same index bits

Associativity: the number of blocks in a set
- direct mapped
- set associative
- fully associative

Size: # bytes of data

How do you calculate this?

Logical Diagram of a Cache

Address (showing bit positions)

Hit Tag

Data

Block offset

16 bits 128 bits

4K entries
**Logical Diagram of a Set-associative Cache**

Accessing a Cache

**General formulas**

- number of index bits = \(\log_2(\text{cache size} / \text{block size})\)
  (for a direct mapped cache)
- number of index bits = \(\log_2(\text{cache size} / (\text{block size} \times \text{associativity}))\)
  (for a set-associative cache)
**Design Tradeoffs**

**Cache size**

- the bigger the cache,
  - the higher the hit ratio
  - the longer the access time

**Block size**

- the bigger the block,
  - the better the spatial locality
  - less block transfer overhead/block
  - less tag overhead/entry (assuming same number of entries)
  - might not access all the bytes in the block
**Design Tradeoffs**

**Associativity**

the larger the associativity,
+ the higher the hit ratio
- the larger the hardware cost (comparator/set)
- the longer the hit time (a larger MUX)
- need hardware that decides which block to replace
- increase in tag bits (if same size cache)

Associativity is more important for small caches than large because more memory locations map to the same line e.g., TLBs!

---

**Memory update policy**

- **write-through**
  - performance depends on the # of writes
  - store buffer decreases this
    - check on load misses
    - store compression
- **write-back**
  - performance depends on the # of dirty block replacements but...
    - dirty bit & logic for checking it
    - tag check before the write
    - must flush the cache before I/O
    - optimization: fetch before replace
  - both use a merging store buffer
Design Tradeoffs

Cache contents
- **separate** instruction & data caches
  - separate access \( \Rightarrow \) double the bandwidth
  - shorter access time
  - different configurations for I & D
- **unified** cache
  - lower miss rate
  - less cache controller hardware

Address Translation

**In a nutshell:**
- maps a virtual address to a physical address, using the page tables
- number of page offset bits = **page size**
**TLB**

**Translation Lookaside Buffer (TLB):**
- cache of most recently translated virtual-to-physical page mappings
- typical configuration
  - 64/128-entry
  - fully associative
  - 4-8 byte blocks
  - .5 - 1 cycle hit time
  - low tens of cycles miss penalty
  - misses can be handled in software, software with hardware assists, firmware or hardware
  - write-back
- works because of locality of reference
- much faster than address translation using the page tables

---

**Using a TLB**

1. Access a 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 page reference bit;
   - if writing, set the page dirty bit.
3. If a miss,
   - get the physical address from the page table;
   - evict a TLB entry & update dirty/reference bits in the page table;
   - update the TLB with the new mapping.
Design Tradeoffs

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
  - compiler support for better data placement

- need to flush the cache on a context switch
  - thread identification (TID) 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 in the cache
    - on a write, only one will be updated; so the other has old data
  - a solution: page coloring
    - processes share segments; all shared data have the same offset from the beginning of a segment, i.e., the same low-order bits
    - cache must be \( \leq \) the segment size
      (more precisely, each set of the cache must be \( \leq \) the segment size)
    - index taken from segment offset, tag compare on segment #
Design Tradeoffs

Virtual or physical addressing

Physically-addressed caches
- access with a physical index & compare with physical tag
- do address translation on every cache access
- no cache flushing on a context switch
- no synonym problem

Physically-addressed caches
- if a straightforward implementation, hit time increases because must translate the virtual address before accessing 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 (virtual & physical bits are the same): virtually indexed
    - access the TLB & cache at the same time
    - compare the physical tag from the cache to the physical address (page frame #) from the TLB: physically tagged
    - can increase cache size by increasing associativity, but still use page offset bits for the index
Cache Hierarchies

Cache hierarchy
- different caches with different sizes & access times & purposes
+ decrease effective memory access time:
  - many misses in the L1 cache will be satisfied by the L2 cache
  - avoid going all the way to memory

Level 1 cache goal: fast access
so minimize hit time (the common case)
Cache Hierarchies

Level 2 cache goal: keep traffic off the system bus

Cache Metrics

Hit (miss) ratio = \frac{\#\text{hits}}{\#\text{misses}}

• measures how well the cache functions
• useful for understanding cache behavior relative to the number of references
• intermediate metric

Effective access time = \text{HitTime} + \text{Miss Ratio} \times \text{Miss Penalty}

• (rough) average time it takes to do a memory reference
• performance of the memory system, including factors that depend on the implementation
• intermediate metric
Measuring Cache Hierarchy Performance

Effective Access Time for a cache hierarchy:

\[
\text{Hit time}_{1,1} + \text{Miss ratio}_{1,1} \times \text{Miss penalty}_{1,1}
\]

\[
\text{Hit time}_{1,2} + \text{Miss ratio}_{1,2} \times \text{Miss penalty}_{1,2}
\]

... 

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 Ratio:**

\[
globalMR = \frac{\text{# misses in cache}}{\text{# references generated by CPU}}
\]

Example: 1000 References
- 40 L1 misses
- 10 L2 misses

**global MR (L1):**

**global MR (L2):**

---

Miss Classification

Usefulness is in providing insight into the causes of misses
- does not explain what caused a particular, individual miss

**Compulsory**
- first reference misses
- decrease by increasing block size

**Capacity**
- due to finite size of the cache
- decrease by increasing cache size

**Conflict**
- too many blocks map to the same set
- decrease by increasing associativity

**Coherence (invalidation)**
- decrease by decreasing block size + improving processor locality