Introduction

Why memory subsystem design is important

- CPU speeds increase 55% per year
- memory speeds increase 7% per year
- rate of increase is also widening

Approaches to decreasing the memory penalty

- eliminate memory operations
- decrease the number of misses
- decrease miss penalty
- decrease cache/memory access times
- hide memory latencies

Introduction

Memory hierarchies

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

Memory hierarchies work!

- principal of **locality of reference**
  - temporal
  - spatial
  - processor
  - value
- technology
  - speed vs. size
- $\Longrightarrow$ fast access for most references

Cache Review

Organization

- Size
- Block
- Set
- Associativity
  - direct-mapped
  - set-associative
  - fully-associative

Assessing a cache

- index = $\log_2$(cache size/block size) (DM)
  $\log_2$(cache size/block size*associativity) (SA)
- tag = word size - index - $\log_2$(block size)

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

Metrics

Hit (miss) ratio

$$\frac{\#\text{hits(misses)}}{\#\text{references}}$$

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

Effective access time

$$\text{Time}_{\text{hit}} + \text{MissRatio} \bullet \text{Time}_{\text{miss}}$$

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

Review of Cache Metrics

Misses per instruction

$$\frac{\text{misses}}{\text{instruction}} = \frac{\text{memory accesses}}{\text{instruction}} \bullet \text{miss rate}$$

- intermediate metric
- hardware-independent, architecture-dependent

Execution time including the memory system

$$\frac{\text{instructions}}{\text{program}} \bullet (\text{CPI}_{\text{CPU}} + \text{CPI}_{\text{memory}}) \bullet \text{cycle time}$$

$$\text{CPI}_{\text{memory}} = \frac{\text{memory accesses} \bullet \text{miss rate} \bullet \text{miss penalty}}{\text{instruction}}$$

- how memory stalls contribute to the performance bottom line

Miss Classification

Compulsory

- increase block size

Capacity

- increase cache size

Conflict

- increase associativity

Coherence

- decrease block size + improve processor locality

Design Tradeoffs

Cache size

- hit ratio vs. access time

The bigger the cache

+ the higher the hit ratio
- the longer the access time

Design Tradeoffs

Block size

the bigger the block

+ the more you can take advantage of spatial locality
+ less block transfer overhead/block
+ less tag overhead/entry
- might not access all the bytes in the block
- false sharing *(later)*

Design Tradeoffs

Associativity

the larger the associativity:

+ the higher the hit ratio
- the larger the hardware cost (comparators)
- the larger the hit time (a larger MUX)
- need block replacement hardware
- increase in tag bits

Why is associativity more important for small caches than large?

Why does 2-way set associative cache have a lower miss ratio than a direct mapped cache of the same size?

Design Tradeoffs

Block replacement

- LRU
- random

How would you implement LRU?

How would you implement random?

When would LRU produce its worse behavior?

Design Tradeoffs

Memory update policy

- **write through**
  - performance depends on the # of writes
  - write buffer
    - read miss check
    - store compression (coalescing write buffers)
- **write back**
  - performance depends on the # of dirty block replacements but...
  - dirty bit
  - tag check before the write
  - flush cache before I/O
  - optimization: fetch before replace

Design Tradeoffs

Cache contents

- **separate** instruction & data caches
  - separate access $\Longrightarrow$ double the bandwidth
  - different configurations for I & D: why?
  - shorter access time
- **unified** cache
  - lower miss rate
  - less cache controller hardware

Design Tradeoffs

Virtual or physical addressing

virtually-addressed caches (virtual tags)

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

\+ faster for hits because no address translation

physically-addressed caches (physical tags)

- access with a physical address (index & tag)
- do virtual-to-physical address translation on every access

\- increase in hit time because must translate the virtual address before access the cache

virtually-addressed caches (physical tags)

- same fast hit time as a virtual cache
- address translation in parallel with cache access