Cache

Philippos Papaphilippou

Introduction

- **Cache** is a fast memory in the CPU designed to store a small amount of data, either duplicates RAM or newly computed values
- When a requested block of data is found in cache it is considered a **hit**
- When there is a need to fetch it from RAM it is considered a **miss**

Hit and miss rates

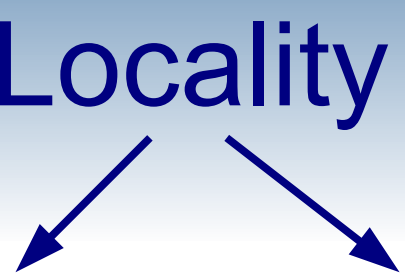
- The higher the requested information is found in cache the higher **hit rate** gets
- To achieve performance, a lot of effort is to minimize miss rate and increase hit rates
- **Miss penalty** is the time required to fetch data from ram on a miss
- Miss penalty and miss rates are the main contributors to lower performance
- The simplest form of handling a miss is to stall

Memory Hierarchy

- Due to performance and cost reasons, memory is organized hierarchically

Technology	Average access time	Average cost per GB ($)	Level
SRAM	3 ns	3000	higher
DRAM	60 ns	50	lower
Hard Drive	10 000 000 ns	1	lower

- The goal is to create the illusion of getting all the data from the highest levels of memory



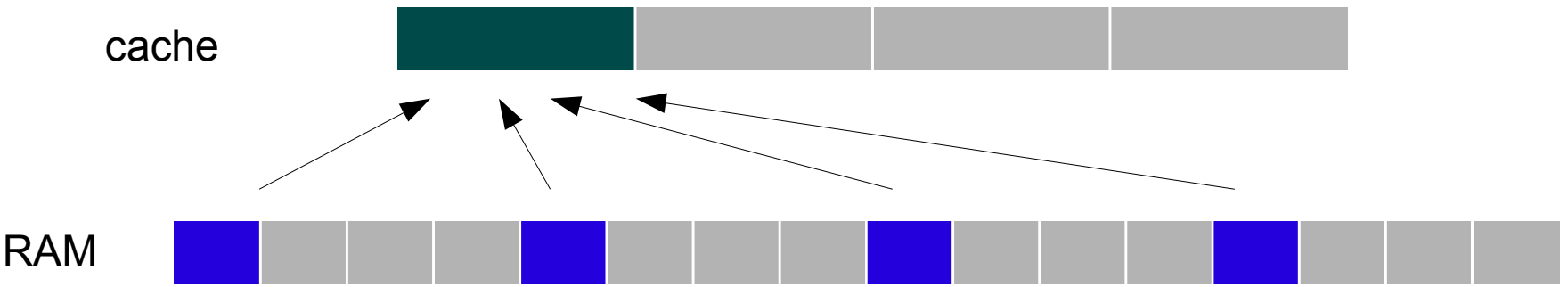
Spatial Temporal

- We try to keep important data in cache using **temporal and spatial localities**
- **Temporal locality** gives priority to recently requested items
- **Spatial locality** gives priority to items whose address is close to a requested item

Direct-Mapped Cache

- It is a basic model of organizing the cache
- Use this formula to find a block

  (Block address) modulo (Number of blocks in cache)
- Many blocks are represented by the same block in cache according to the last bits of the address in order to correspond with cache's addresses



Direct-Mapped Cache

- Advantages:

  A corresponding cache block for every DRAM block

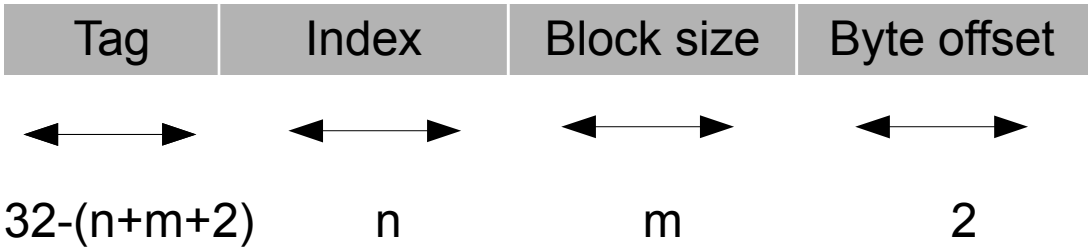
  Temporal locality

- Disadvantage:

  The DRAM blocks represented by the same cache block can not coexist in cache

Number of bits in Cache

- Using a 32-bit Address



- We have 2^n cache addresses of block size data in bits (2^{m+5}), tag size (32-n-m-2) and a bit for valid tag

Number of bits = 2^n * (2^m *32 +31 -n -m)

Write Schemes

- **Write-through** stalls the processor for each block change to copy data into both cache and DRAM
- Write buffers can speed the process
- **Write-back** writes directly into cache and copies new data to DRAM as soon as the changed blocks in cache need to be replaced
- Write-back is faster and more complicated to implement than write-through
- The blocks in DRAM which are out of date are called **dirty blocks**

Faster Implementations

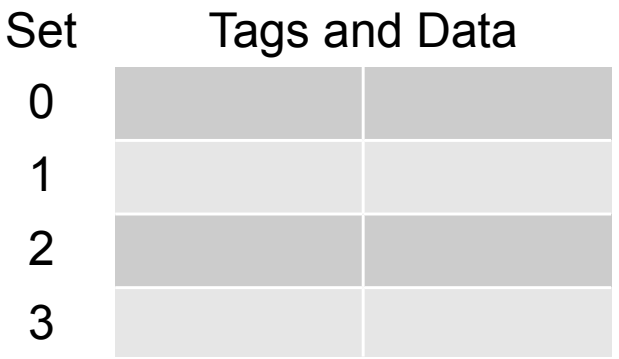
- Splitting the cache into instructions and data caches can give greater performance over a combined cache

  (Doing so doubles the bandwidth and overcomes a minor increment in miss rate)
- Widening the cache, memory and interconnection bus
- Using interleaved memory organization to allow simultaneous reads from cache
- DDR (Double data rate)

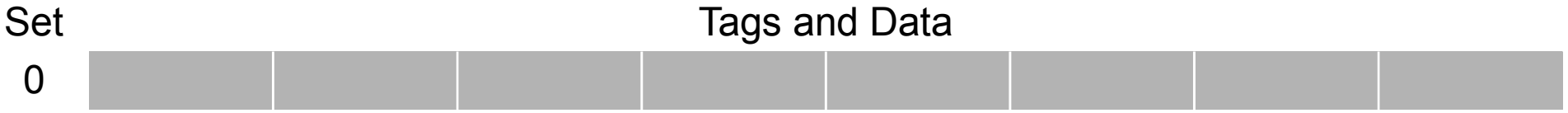
Associative Cache

- A **set associative** cache allows memory blocks to be placed to n locations in one set in cache
- two-way associative example

  (for 8 locations)



- **Fully associative** cache allows memory blocks to be placed anywhere and therefore tag includes the whole address (minus offsets) of the stored block



(8 way cache)

Associative Cache

- Therefore direct-mapped cache can be considered as one-way cache
- **Replacement policies** need to be applied when replacing a block in associative cache because there is a choice of n elements to be removed
- A common policy is to replace the **least recently used (LRU)** element in a set
- Each block in main memory is directly mapped to a set and the set in which it may be found is

  (Address) modulo (number of sets)

Comparison of Associative Cache

- The bigger the degree of associativity cache has
  - miss rate decreases (localities) ✔
  - tag field increases ✘
  - latency increases (longer searching) ✘
  - Implementation of LRU gets harder ✘
  - production cost increases ✘

- Therefore there are many factors to consider for the degree of associativity of a cache (e.g. size)

Multilevel Caches

- Today we use multilevel caches to reduce miss penalty
- There are primary and second-level caches

  - Primary cache is the smallest memory and has high miss rate

  - Second-level is bigger than the size of one cache implementation and has smaller access time than main memory
- While there is complexity and hardware cost limits, this **memory hierarchy scheme** improves performance significantly

Multilevel Caches

- Secondary cache has larger block-size and associativity as it is critical for miss rates
- The **global miss rate** refers to the miss rate of all levels of cache
- The **local miss rate** refers to the miss rate of a single level in cache
- While the local miss rate for a secondary cache may be very large, global miss rate is responsible for accesses to the main memory

Measuring Time

- We can compute the overall CPU time with the above formula

  CPU time = (CPU execution clock cycles
  + Memory-stall clock cycles) * Clock period

  where Memory-stall clock cycles =
  Read-stall cycles + Write-stall clock cycles

  where Read/Write-stall cycles =
  (Read/Writes / Program * miss rate * miss penalty)
  + Write buffer stalls

  or Memory-stall clock cycles =

$$\frac{Memory\ accesses}{Program} * Miss\ rate * Miss\ penalty = \frac{Instructions}{Program} * \frac{Misses}{Instruction} * Miss\ penalty$$

Measuring Time

- For out-of-order processors

  Memory-stall clock cycles =
  Misses * (Total miss latency – Overlapped miss latency)

- Average memory access time

  AMAT = Time for a hit + Miss rate * Miss penalty

- Effective CPI can be found using

  CPI = Base CPI + Memory-stall clock cycles

  where Memory-stall clock cycles is the sum of stalls per instruction for every level if cache is multilevel

- And for measuring performance

$$\frac{CPU\ time\ with\ stalls}{CPU\ time\ with\ perfect\ cache} = \frac{I * CPI\,(stall) * Clock\ cycle}{I * CPI\,(perfect) * Clock\ cycle} = \frac{CPI\,(stall)}{CPI\,(perfect)}$$

Algorithms Performance

- The memory hierarchy is often ignored for software development
- But, it is important to understand the impact of cache design to improve cache utilization and therefore performance
- There have been studies for choosing different sorting algorithms (e.g. Bubble sort) to show benefits over higher number of instructions, clock cycles and cache misses per item
- When the attributes for faster software implementations are very complex, **autotuning** is used where the computer tests access times and chooses the best implementation

The End

Source

- Computer Organization and Design 4th Edition David A. Patterson, John L. Hennessy

http://www.cs.ucy.ac.cy/~ppapap01/

Xi Lab University of Cyprus 2013

Philippos Papaphilippou