CS4617 Computer Architecture

Lecture 3: Memory Hierarchy 1

Dr J Vaughan

September 15, 2014

Important terms

Cache	fully associative	write allocate
Virtual memory	dirty bit	unified cache
Memory stall cycles	block offset	misses per instruction
Direct mapped	write-back	block
Valid hit	data cache	locality
Block address	hit time	address trace

Table: Memory terms

Important terms (ctd)

Write-through	Cache miss	Set
Instruction cache	Page fault	Random replacement
Average memory access time	Miss rate	Index field
Cache hit	n-way set associative	No-write allocate
Page	Least recently used	Write buffer
Miss penalty	Tag field	Write stall

Table: Memory terms

Term definitions 1

Cache The first level of memory met when the address leaves the processor. The word cache is often used to mean buffering commonly-used items for re-use

Cache hit Success: finding a referenced item in cache

Cache miss Failure: the required item is not in the cache

Block The fixed number of bytes of main memory that is copied to the cache in one transfer operation. This transfer happens when a cache miss occurs

Temporal locality A referenced item is likely to be referenced again in the near future

Spatial locality Other data in the same block as a referenced item are likely to be needed soon

Term definitions 2

Virtual memory The extension of memory to an address space that encompasses the physical residence on disk of some programs and data

Page A fixed-size block of virtual memory address space, usually in the range 1K to 4K.
A page is either in main memory or on disk

Page fault An interrupt generated when the processor references a page that is neither in cache nor in main memory

Term definitions 3

Memory stall cycles Number of cycles during which processor is stalled waiting for a memory access

Miss penalty Cost per cache miss

Address trace A record of instruction and data references with a count of the number of accesses and miss totals

Term definitions 4

- $CPU\ execution\ time =$
  $(CPU\ clock\ cycles + Memory\ stall\ cycles) \times clock\ cycle\ time$
  Assumes CPU clock cycles include time to handle a cache hit and that the processor is stalled during a cache miss
- $Memory\ stall\ cycles = Number\ of\ misses \times Miss\ penalty$
  $= IC \times \frac{Misses}{Instruction} \times Miss\ penalty$
  $= IC \times \frac{Memory\ accesses}{Instruction} \times Miss\ rate \times Miss\ penalty$
  where IC = instruction count
- Miss rate
  $Fraction\ of\ cache\ accesses\ that\ miss =$
  $\frac{Number\ of\ accesses\ that\ miss}{Number\ of\ accesses}$
- Miss rates and miss penalties are different for reads and writes but are averaged here

Example

- Computer cycles per instruction (CPI), is 1.0 when all memory accesses are cache hits. The only data accesses are loads and stores, representing a total of 50% of the instructions.
  If the miss penalty is 25 clock cycles and the miss rate is 2%, how much faster would the computer be if all instructions were cache hits?

Solution

If all accesses are cache hits:

- $CPU\ execution\ time =$
  $(CPU\ clock\ cycles + Memory\ stall\ cycles) \times Clock\ cycle =$
  $(IC \times CPI + 0) \times Clock\ cycle = IC \times 1.0 \times Clock\ cycle$

With real cache:

- $Memory\ stall\ cycles =$
  $IC \times \frac{Memory\ accesses}{Instruction} \times Miss\ rate \times Miss\ penalty$
  $= IC \times (1 + 0.5) \times 0.02 \times 25 = IC \times 0.75$
  where $(1 + 0.5)$ represents 1 instruction access and 0.5 data accesses per instruction

Solution (continued)

- Total performance:

$$\begin{aligned}
&\textit{CPU execution time}_{\textit{cache}} \\
&= (\textit{IC} \times 1.0 + \textit{IC} \times 0.75) \times \textit{Clock cycle time} \\
&= 1.75 \times \textit{IC} \times \textit{Clock cycle time} \\
&\textit{Performance ratio} = \frac{\textit{CPU execution time}_{\textit{cache}}}{\textit{CPU execution time}} \\
&= \frac{1.75 \times \textit{IC} \times \textit{Clock cycle time}}{1.0 \times \textit{IC} \times \textit{Clock cycle time}} \\
&= 1.75
\end{aligned}$$

- So computer with no cache misses is 1.75 times faster

Misses per instruction

- $\frac{Misses}{Instruction} = \frac{Miss\ \ rate \times Memory\ \ accesses}{Instruction\ \ count}$
  $= Miss\ \ rate \times \frac{Memory\ \ accesses}{Instruction}$
- This formula is useful when the average number of memory accesses per instruction is known
  It allows conversion of miss rate into misses per instruction and vice versa
- In the last example,
  $\frac{Misses}{Instruction}$
  $= Miss\ \ rate \times \frac{Memory\ \ accesses}{Instruction}$
  $= 0.02 \times 1.5 = 0.03$

Example

- Same data as previous example
- What is memory stall time in terms of instruction count?
- Assume miss rate of 30 per 1000 instructions

Answer

- Memory stall cycles
  $= \textit{Number of misses} \times \textit{Miss penalty}$
  $= IC \times \frac{\textit{Misses}}{\textit{Instruction}} \times \textit{Miss penalty}$
  $= IC \times 0.75$

Four Memory Hierarchy Questions

Q1: Block placement Where can a block be placed in the upper level?

Q2: Block identification How is a block found if it is in the upper level?

Q3: Block replacement Which block should be replaced on a miss?

Q4: Write strategy What happens on a write?

Q1: Where can a block be placed in cache?

Three organisations

Direct mapping Line = (Block address) *mod* (Number of blocks in cache)

Associative mapping Block can be placed in any line

Set-associative mapping n lines per set = n-way set
Set = block address mod Number of sets in cache
Place block in any set line

Mapping

- Direct mapping = 1-way set-associative
- Associative with m blocks = m-way set associative
- Most processor caches are either
  - Direct mapped
  - 2-way set associative
  - 4-way set associative

Q2: How is a block found if it is in cache?

- A tag on every block frame gives the block address
- All possible tags are searched in parallel for tag of required block
- A valid bit used to indicate if block contents are valid
- If the valid bit is not set, there is no match

Memory address from processor

- $Address =< Block\ address >< Block\ offset >$
- $Block\ address =< Tag >< Index >$
- Index field selects set
- Tag field used to search in set for a hit
- Offset selects data when block found

Q3: Which block should be replaced on a cache miss?

- Direct mapping
  - Only 1 block frame checked for a hit, only that block can be replaced
- Fully associative or set associative
  - Choice of which block to replace

Replacement strategies

- Random
  - Selects block for replacement randomly
- LRU
  - Relies on locality
- FIFO
  - LRU is difficult to calculate so the oldest block is selected for replacement

Q4: What happens on a write?

- Reads dominate processor cache access
- All instruction fetches are reads
- Most instructions do not write to memory
- Make common case fast
  - Optimize for reads

Common case is easy to make fast

- Read block from cache at the same time that the tag is read and compared
- Block read begins as soon as block address is available
- If read is a hit, requested part of block is passed to CPU immediately
- If read is a miss, no benefit, no harm, just ignore the value read

Write

- Tag checking and block modification cannot occur in parallel
- Therefore, writes take longer than reads
- Processor specifies size of write (between 1 and 8 bytes) so only that part of the block can be changed
- Reads can access more bytes than necessary without difficulty

Write policies

- Write-through
  - Information written to the block in cache and to lower-level memory
- Write-back
  - Only write to block in lower-level memory if dirty bit set when block is replaced

Advantages of write-back

- Writes occur at the speed of cache memory
- Multiple writes within a block require only 1 write to lower-level memory
- So write-back uses less memory bandwidth which is useful in multiprocessors
- Write-back uses the memory hierarchy and interconnect less than write-through so it saves power and is appropriate for embedded applications

Advantages of write-through

- Easier to implement than write-back
- Cache is always clean so misses never cause a write to the lower level
- Next lower level has current copy of data which simplifies data coherence
- Data coherence is important for multiprocessors and I/O
- Multilevel caches make write-through more viable for the upper-level caches as the writes need only propagate to the next lower level rather than all the way to main memory