CS4617 Computer Architecture

Lecture 5: Memory Hierarchy 3

Dr J Vaughan

September 22, 2014

Six basic cache optimisations

$$Average\ memory\ access\ time = Hit\ time + Miss\ rate \times Miss\ penalty$$

Thus, cache optimisations can be divided into 3 categories

Reduce the miss rate Larger block size, larger cache size, higher associativity

Reduce the miss penalty Multilevel caches, give reads priority over writes

Reduce the time for a cache hit Avoid address translation when indexing the cache

Reducing cache misses

All misses in single-processor systems can be categorised as:

Compulsory
: The first access to a block cannot be in cache
  - Called a *cold-start miss* or *first-reference miss*

Capacity
: Misses due to cache not being large enough to contain all blocks needed during execution of a program

Conflict
: In set-associative or direct mapped organisations, conflict misses occur when too many blocks are mapped to the same set, leading to some blocks being replaced and later retrieved
  - Also called collision misses
  - Hits in an associative cache that become misses in an n-way set-associative cache are due to more than n requests on some high-demand sets

Conflict miss categories

Conflict misses can be further classified in order to emphasise the effect of decreasing associativity

Eight-way
: Conflict misses due to going from fully-associative (0 conflicts) to 8-way associative

Four-way
: Conflict misses due to going from 8-way to 4-way associative

Two-way
: Conflict misses due to going from 4-way to 2-way associative

One-way
: Conflict misses due to going from 2-way associative to direct mapping

Conflict misses

- In theory, conflicts are the easiest problem to solve
- Fully associative organisation prevents all conflict misses
- However, this may slow the CPU clock rate, lead to lower overall performance and is expensive in hardware
  (Why is this?)
- Capacity has to be addressed by increasing cache size
- If upper-level memory is too small, time is wasted in moving blocks to and fro between the two memory levels – *thrashing*

Comments on the 3-C model

- The 3-C model gives insight into average behaviour
- Changing cache size changes conflict misses as well as capacity misses, since a larger cache spreads out references to more blocks
- The 3-C model ignores replacement policy: it is difficult to model and generally less significant
- In certain circumstances the replacement policy can lead to anomalous behaviour such as poorer miss rates for larger associativity
  Relate to replacement in demand paging: Belady's anomaly – does not occur with stack algorithms
- Many techniques that reduce miss rates also increase hit time or miss penalty

First Optimisation: larger block size to reduce miss rate

Q: How does larger block size reduce miss rate?
A: Locality $\implies \uparrow$number of 'working set' elements available in cache

- There is a trade-off between block size and miss rate
- Larger blocks take advantage of spatial locality
- Larger blocks also reduce compulsory misses
  Because for fixed cache size, #blocks$\downarrow$ as block size$\uparrow$
- But larger blocks increase the miss penalty
- The increase in miss penalty may outweigh the decrease in miss rate

Example

- Memory system takes 80 clock cycles of overhead and then delivers 16 bytes every 2 clock cycles.
- Referring to this table, which block size has the smallest average memory access time for each cache size?

	Cache sizes			
Block size (kB)	**4K (%)**	**16K (%)**	**64K (%)**	**256K (%)**
16	8.57	3.94	2.04	1.09
32	7.24	2.87	1.35	0.7
64	7.0	2.64	1.06	0.51
128	7.78	2.77	1.02	0.49
256	9.51	3.29	1.15	0.49

Table: Miss rate vs block size for different-sized caches (Fig B.11, H&P)

Example (continued)

$$\textit{Average memory access time} = \textit{Hit time} + \textit{Miss rate} \times \textit{Miss penalty}$$

- Assume hit time is 1 clock cycle independent of block size
- Recall from problem statement: 80 clock cycles of overhead and then 16 bytes every 2 clock cycles
- 16-byte block, 4KB cache

$$\textit{Average memory access time} = 1 + .0857 \times 82 = 8.027 \textit{ clock cycles}$$

- 256-byte block, 256KB cache

$$\textit{Average memory access time} = 1 + .0049 \times 112 = 1.549 \textit{ clock cycles}$$

Example (continued)

Block size (kB)	**Miss penalty** (clock cycles)	**Cache sizes**			
		4K (%)	**16K (%)**	**64K (%)**	**256K (%)**
16	82	8.027	4.231	2.673	1.894
32	84	**7.082**	3.411	2.134	1.588
64	88	7.16	**3.323**	**1.933**	**1.449**
128	96	8.469	3.659	1.979	1.47
256	112	11.651	4.685	2.288	1.549

Table: Mean memory access time vs block size for different-sized caches (Fig B.12, H&P 5e)

Optimization 2: Larger caches to reduce miss rate

$$\uparrow\text{Cache size} \implies \uparrow\text{Prob(referenced word in cache)} \implies \downarrow\text{Miss rate}$$

- Possible longer hit time
  1. As cache size $\uparrow$, time to search cache for a given address $\uparrow$
  2. As cache size $\uparrow$, it may be necessary to place cache off-chip
- Possible higher cost and power
- Popular in off-chip caches

Optimization 3: Higher associativity to reduce miss rate

- 8-way set associative is as effective in reducing misses as fully associative
- 2:1 cache rule of thumb
  - A direct mapped cache of size N has about the same miss rate as a 2-way set-associative cache of size $N/2$
- Increasing block size decreases miss rate ($\because$ locality) and increases miss penalty ($\because$ $\uparrow$time to transfer larger block)
- Increasing associativity may increase hit time ($\because$ H/W for parallel search increases in complexity)
- Fast processor clock cycle encourages simple cache designs

Example

- Assume that higher associativity would increase clock cycle time:
  - $\textit{Clock cycle time}_{2-way} = 1.36 \times \textit{Clock cycle time}_{1-way}$
  - $\textit{Clock cycle time}_{4-way} = 1.44 \times \textit{Clock cycle time}_{1-way}$
  - $\textit{Clock cycle time}_{8-way} = 1.52 \times \textit{Clock cycle time}_{1-way}$
- Assume hit time = 1 clock cycle
- Assume miss penalty for direct mapped cache = 25 clock cycles to a L2 cache that never misses
- Assume miss penalty need not be rounded to an integral number of clock cycles

Example (continued)

Under the assumptions just stated:

For which cache sizes are the following statements regarding average memory access time (AMAT) true?

- $AMAT_{8-way} < AMAT_{4-way}$
- $AMAT_{4-way} < AMAT_{2-way}$
- $AMAT_{2-way} < AMAT_{1-way}$

Answer

- $Average\ memory\ access\ time_{8-way}$
  $= Hit\ time_{8-way} + Miss\ rate_{8-way} \times Miss\ penalty_{8-way}$
  $= 1.52 + Miss\ rate_{8-way} \times 25\ clock\ cycles$
- $Average\ memory\ access\ time_{4-way}$
  $= 1.44 + Miss\ rate_{4-way} \times 25\ clock\ cycles$
- $Average\ memory\ access\ time_{2-way}$
  $= 1.36 + Miss\ rate_{2-way} \times 25\ clock\ cycles$
- $Average\ memory\ access\ time_{1-way}$
  $= 1.00 + Miss\ rate_{1-way} \times 25\ clock\ cycles$

Answer (continued)

Using miss rates from Figure B.8, Hennessy & Patterson:

- $\textit{Average memory access time}_{1-way} = 1.00 + 0.098 \times 25 = 3.44$ for a 4KB direct-mapped cache
- $\textit{Average memory access time}_{8-way} = 1.52 + 0.006 \times 25 = 1.66$ for a 512KB 8-way set-associative cache

Note from the table in Hennessy & Patterson Figure B.13 that, beginning with 16KB, the greater hit time of larger associativity outweighs the time saved due to reduction in misses

Associativity example: table from H & P Figure B.13

	Associativity			
Block size (kB)	**1-way**	**2-way**	**4-way**	**8-way**
4	3.44	3.25	3.22	**3.28**
8	2.69	2.58	2.55	**2.62**
16	2.23	**2.4**	**2.46**	**2.53**
32	2.06	**2.3**	**2.37**	**2.45**
64	1.92	**2.14**	**2.18**	**2.25**
128	1.52	**1.84**	**1.92**	**2.0**
256	1.32	**1.66**	**1.74**	**1.82**
512	1.2	**1.55**	**1.59**	**1.66**

Table: Memory access times for k-way associativities. Boldface signifies that higher associativity *increases* mean memory access time

Optimization 4: Multilevel caches to reduce miss penalty

- Technology has improved processor speed at a faster rate than DRAM
- Relative cost of miss penalties increases over time
- Two options:
  - Make cache faster?
  - Make cache larger?
  - Do both by adding another level of cache
- L1 cache fast enough to match processor clock cycle time
- L2 cache large enough to intercept many accesses that would go to main memory otherwise

Memory access time

- Average memory access time = $Hittime_{L1} + Miss\ rate_{L1} \times Miss\ penalty_{L1}$
- $Miss\ penalty_{L1} = Hit\ time_{L2} + Miss\ rate_{L2} \times Miss\ penalty_{L2}$
- Average memory access time = $Hit\ time_{L1} + Miss\ rate_{L1} \times (Hit\ time_{L2} + Miss\ rate_{L2} \times Miss\ penalty_{L2})$

where $Miss\ rate_{L2}$ is measured in relation to requests that have already missed in L1 cache

Definitions

- $Local\ miss\ rate = \frac{Number\ of\ cache\ misses}{Total\ accesses\ to\ this\ cache}$

  For example

  $Miss\ rate_{L1} = \frac{\#\ L1\ cache\ misses}{\#\ accesses\ from\ CPU}$

  $Miss\ rate_{L2} = \frac{\#\ L2\ cache\ misses}{\#\ accesses\ from\ L1\ to\ L2}$

- Global miss rate $=$

  $\frac{Number\ of\ misses\ in\ a\ cache}{Total\ number\ of\ memory\ accesses\ from\ the\ processor}$

  For example

  At L1, global miss rate $= Miss\ rate_{L1}$

  At L2, global miss rate $= Miss\ rate_{L1} \times Miss\ rate_{L2}$

  $\#\ L1\ cache\ misses = \#\ accesses\ from\ L1\ to\ L2$

  $\implies Miss\ rate_{L1} \times Miss\ rate_{L2} = \frac{\#\ L2\ cache\ misses}{\#\ accesses\ from\ CPU}$

- The local miss rate is large for L2 cache because the L1 cache has dealt with the most local references
- Global miss rate may be more useful in multilevel caches

Memory stalls

Average memory stall time per instruction

$= Misses\ per\ instruction_{L1} \times Hit\ time_{L2}$

$+ Misses\ per\ instruction_{L2} \times Miss\ penalty_{L2}$

Example: memory stalls

- 1000 memory references, 40 L1 misses, 20 L2 misses
- What are the miss rates?
- Assume L2 miss penalty is 200 clock cycles
- $Hit\ time_{L2} = 10$ clock cycles
- $Hit\ time_{L1} = 1$ clock cycle
- 1.5 memory references per instruction
- Ignore writes
- What is average memory access time?
- What is average stall cycles per instruction?

Answer: memory stalls

- $Miss\ rate_{L1} = 40/1000 = 4\%$
- $Miss\ rate_{L2} = 20/40 = 50\%$
- $Global\ miss\ rate_{L2} = 20/1000 = 2\%$

$$Average\ memory\ access\ time = 1 + 4\% \times (10 + 50\% \times 200)$$
$$= 1 + 4\% \times 110$$
$$= 5.4\ clock\ cycles$$

Answer: memory stalls (continued)

1000 memory references, 1.5 references per instruction

$\implies$ 667 instructions

$Misses \times 1.5 = Misses\ per\ 1000\ instructions$

$40 \times 1.5 = 60\ L1\ misses\ per\ 1000\ instructions$

$20 \times 1.5 = 30\ L2\ misses\ per\ 1000\ instructions$

Average memory stalls per instruction

$= Misses\ per\ instruction_{L1} \times Hit\ time_{L2}$

$+ Misses\ per\ instruction_{L2} \times Miss\ penalty_{L2}$

$= (60/1000) \times 10 + (30/1000) \times 200$

$= 6.6\ clock\ cycles$

assuming misses are distributed uniformly between instructions and data

Subtracting $Hit\ time_{L1}$ from average memory access time and multiplying by the average number of memory references per instruction gives the same memory stall result:

$(5.4 - 1.0) \times 1.5 = 4.4 \times 1.5 = 6.6\ clock\ cycles$

All these formulae are for combined reads and writes, assuming a write-back L1 cache

Effect of write-through cache

- A write-through L1 cache sends *all* writes to L2, not just the misses
- Miss rates and relative execution time change with the size of L2 cache

1. Global cache miss rate is similar to L2 miss rate, provided that $|L2\ cache| >> |L1\ cache|$
2. Local cache miss rate is not a good measure of L2 caches. $Miss\ rate_{L2}$ is $f(Miss\ rate_{L1})$ and will be varied by changing the L1 cache. Use the global cache miss rate to evaluate L2 cache

Parameters of L2 caches

- Speed of L1 cache affects processor clock rate
- Speed of L2 cache affects $Miss\ penalty_{L1}$

L2 questions:

- Will L2 cache lower the average memory access time part of CPI?
- How much does it cost?
- What should the size of L2 cache be?
  - Everything in L1 is likely to be in L2 also $\implies |L2| >> |L1|$
  - If $|L2|$ just a little bigger than $|L1|$, the local miss rate, $Miss\ rate_{L2}$ will be high
  - Does set associativity make sense for L2 caches?

Example

- Impact of L2 cache associativity on $Miss\ penalty_{L2}$
- $Hit\ time_{L2}$ for direct mapping = 10 clock cycles
- 2-way set associativity increases hit time by 0.1 clock cycles to 10.1 clock cycles ($\because$ $\uparrow$circuit complexity)
- Local $Miss\ rate_{L2}$ for direct mapping = 25%
- Local $Miss\ rate_{L2}$ for 2-way set associativity = 20%
- $Miss\ penalty_{L2}$ = 200 clock cycles

Answer

For a direct-mapped L2 cache,
$Miss\ penalty_{1-way\ L2} = 10 + 0.25 \times 200$
$= 60$ clock cycles
$Miss\ penalty_{2-way\ L2} = 10.1 + 0.2 \times 200$
$= 50.1$ clock cycles
In practice, L2 caches are usually synchronized with the processor and L1 cache.
Thus, $Hit\ time_{L2}$ must be an integral number of cycles: 10 or 11 clock cycles in this example
$Miss\ penalty_{2-way\ L2} = 10 + 0.2 \times 200$
$= 50$ clock cycles
or
$Miss\ penalty_{2-way\ L2} = 11 + 0.2 \times 200$
$= 51$ clock cycles
So $Miss\ penalty_{L2}$ can be reduced by reducing $Miss\ rate_{L2}$

Inclusion and exclusion

- Are L1 data in the L2 cache?
  - *Multilevel inclusion*
  - L1 data are always in the L2 cache
- Inclusion is desirable because consistency between I/O and caches can be checked just by examining L2 cache. If there are smaller blocks for a smaller L1 cache and larger blocks for a larger L2 cache (as in the Pentium 4: 64 byte/128 byte), then inclusion can be maintained with more work on an L2 miss:
  - L2 cache must invalidate all L1 blocks that map onto the L2 block to be replaced
  - This causes a higher L1 miss rate
- Because of this complexity, many designers keep the block size the same at all cache levels

Exclusion

- If L2 cache is only slightly bigger than L1 cache, use *Multilevel exclusion*
- L1 data *never* in L2
- *Cache* $miss_{L1}$ causes a swap of blocks between L1 and L2 instead of replacement
- This prevents wasting space in L2 cache
- Example: AMD Opteron
- L1 cache design is simpler if there is a compatible L2 cache

Example

- Write-through at L1 is less risky (in terms of time penalty) if there is write-back at L2 (to reduce the cost of repeated writes) and multilevel inclusion is used
- Cache design: balance fast hits and few misses
- L2 cache: hit rate lower than L1
- L2 cache: concentrate on fewer misses
- This leads to larger caches and techniques to reduce the miss rate, such as higher associativity and larger blocks

Optimization 5: Give priority to read misses over writes to reduce the miss penalty

- Serve reads before writes have been completed
- Consider the complexities of a write buffer
- Write buffer of appropriate size is important for write-through cache
- Memory access is complicated because the write buffer may hold the updated value of a location needed on a read miss

Example

$SW\ \ R3, 512(R0); M[512] \leftarrow R3\ (cache\ \ index\ 0)$
$LW\ \ R1, 1024(R0); R1 \leftarrow M[1024]\ (cache\ \ index\ 0)$
$LW\ \ R2, 512(R0); R2 \leftarrow M[512]\ (cache\ \ index\ 0)$

- Assume direct-mapped write-through cache that maps 512 and 1024 to the same block
- Assume 4-word write-through buffer, not checked on a read miss
- Is value in R2 always equal to value in R3?

Answer

This is a read-after-write data hazard
$M[512] \leftarrow R3$;(cache index 0) Writes to write buffer before M[512]
$R1 \leftarrow M[1024]$;(cache index 0) Cache miss because of direct mapping
$R2 \leftarrow M[512]$;(cache index 0) Cache miss: loads R2 from L2 cache
But L2 may not have been updated from the write buffer at the time that the Load R2 instruction is executed
Approaches to dealing with read-after-write data hazard

1. Read miss waits until write buffer is empty
2. Check write buffer contents on a read miss. If there are no conflicts and memory is available, let the read miss continue

All desktop and server processes use approach 2 and give reads priority over writes

Reducing costs with write-back cache

If a read miss causes replacement of a dirty block

- Normally, write out dirty block, than read memory
- Instead, copy dirty block to buffer, read memory, write out dirty block
- This reduces processor waiting time on read
- Allowance must also be made for read-after-write data hazard

Optimization 6: Avoid address translation during indexing of cache to reduce hit time

- Hit time can affect processor clock rate
- Even in processors that take several cycles to access cache, cache access time can limit clock cycle rate
- Cache must deal with translation of virtual address from processor to physical memory address
- Make common case fast $\implies$ use virtual addresses for cache $\rightarrow$ *virtual cache*
- Cache that uses physical addresses $\rightarrow$ *physical cache*

Cache hit time

- There are two tasks:
  - Index the cache
  - Compare addresses
- Index the cache: physical or virtual address?
- Tag comparison: physical or virtual address?
- Full virtual addressing for both indices and tags eliminates translation time from a cache hit
- However, potential problems with virtual cache are:
  - Page-level protection is part of virtual to physical address translation
  - Cache flushing on process switch because virtual addresses are different to physical addresses
  - Aliasing due to different processes using different virtual addresses for the same physical address