

Lecture 14 Memory Hierarchy (2)

Instructor: Jun Yang

Recall the cache design issues

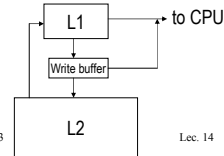
- Main design issues
 - block identification
 - ◊ which is the right block
 - block placement
 - ◊ where to put the new block
 - block replacement
 - ◊ which block to evict
 - write strategy
 - ◊ update L1 only, or both L1 & L2
 - allocation strategy
 - ◊ what to do on a *write miss*?
 - block transfer
 - ◊ fetch the whole block first, or the requested word – critical word first (pp.14).
 - consistency
 - ◊ what about the DMA?
 - ◊ see write strategy

Write Policies

- Writes are hard
 - read: concurrently check tag and read data
 - write is destructive, so it must be slower
- Write strategies
 - when to update memory?
 - ◊ on every write (*write-through*)
 - ◊ when a modified block is replaced (*write-back*)
 - what to do on a *write miss*?
 - ◊ fetch the block to cache (*write allocate*), used with write-back
 - ◊ do not fetch (*write around*), used with write-through
- Trade-offs
 - Write-back:
 - ◊ uses bus bandwidth more efficiently
 - ◊ saves power since it uses lower-level memory less often
 - ◊ attractive to both multiprocessors and embedded applications.
 - Write-through:
 - ◊ Easier to implement
 - ◊ keeps MM consistent with CM,
 - ◊ Also good for multiprocessors since data is coherent in the mem hierarchy.

Write Buffers

- Write Buffers in write-through caches (for bandwidth)
 - buffers words to be written in L2 cache/memory along with their addresses.
 - 2 to 4 entries deep
 - all read misses are checked against pending writes for dependencies (associatively)
 - can coalesce writes to same address, called write merging (pp 15 and 16)
- Write-back Buffers (for speed)
 - between a write-back cache and L2 or MM
 - algorithm
 - ◊ move dirty block to write-back buffer
 - ◊ read new block
 - ◊ write dirty block in L2 or MM
 - can be associated with victim cache (later)



Cache Performance

- Different measure: AMAT
- Average Memory Access time (AMAT) = Hit Time + (Miss Rate x Miss Penalty) = HitTime*HitRate + MissTime*MissRate
(MissTime = Hit Time + Miss Penalty)
- CPU time = (CPU execution clock cycles w/o memory stalls + Memory stall clock cycles) x clock cycle time
- Note: *memory hit time is included in execution cycles.*
- CPI = CPI_{ideal} + Memory stall per instruction

Impact on Performance

- Suppose a processor executes at
 - Clock Rate = 200 MHz (5 ns per cycle)
 - Base CPI = 1.1
 - 50% arith/logic, 30% ld/st, 20% control
 - Hit in the cache takes 1 cycle
- Suppose that 10% of data (ld/st) operations get 50 cycle miss penalty
- Suppose that 1% of instructions get same miss penalty
- CPI = Base CPI + average stalls per instruction

$$= 1.1(\text{cycles/ins}) + [0.30(\text{DataMops/ins}) \times 0.10(\text{miss/DataMop}) \times 50(\text{cycle/miss})] + [1(\text{InstMop/ins}) \times 0.01(\text{miss/InstMop}) \times 50(\text{cycle/miss})]$$

$$= (1.1 + 1.5 + .5) \text{ cycle/ins} = 3.1$$
- AMAT_{program} = I-cache access% x AMAT_{instruction} + D-cache access% x AMAT_{data} = (1/1.3)x[1+0.01x50]+(0.3/1.3)x[1+0.1x50]=2.54

Implications

$$\text{CPU time} = (\text{CPU execution clock cycles} + \text{Memory stall clock cycles}) \times \text{clock cycle time}$$

$$= (\text{CPU}_{\text{exe}} + \text{ms}) \times \text{cc}$$

- When ms and cc are fixed
 - Smaller CPU_{exe} (e.g. better scheduling), larger impact of ms on CPU time.
- When memory hierarchy and structures are identical (ms * cc same, i.e., cache miss penalty is the same)
 - Higher clock rate (frequency), lower cc, higher ms. If the pipeline (CPU_{exe}) is kept constant, higher clock rate has larger impact of memory penalty.
- When CPU_{exe} is fixed
 - cc is affected by the cache structure. Higher associativity leads to higher cc but lower miss rates (and lower ms).
 - Do we gain or lose? Or what is the tradeoff between associativity and cc?

11/21/2003

Lec. 14

7

Impact of Change in cc

- Suppose a processor has the following parameters:
 - CPI = 2 (w/o memory stalls)
 - mem access per instruction = 1.5
- Compare AMAT and CPU time for a direct mapped cache and a 2-way set associative cache assuming:

	cc	Hit cycle	Miss penalty	Miss rate
Direct map	1ns	1	75 ns	1.4%
2-way associative	1.25ns(why?)	1	75 ns	1.0%

- AMAT_d = hit time + miss rate * miss penalty = 1*1 + 0.014*75 = 2.05 ns
- AMAT₂ = 1*1.25 + 0.01*75 = 2 ns < 2.05 ns
- CPU_d = (CPI*cc + mem. stall time)*IC = (2*1 + 1.5*0.014*75)IC = 3.575*IC
 - CPU₂ = (2*1.25 + 1.5*0.01*75)IC = 3.625*IC > CPU_d!
- Change in cc affects all instructions while reduction in miss rate benefit only memory instructions.

11/21/2003

Lec. 14

8

Miss Penalty for Out-of-Order Exe. Proc.

- In OOO processors, memory stall cycles are overlapped with execution of other instructions. Miss penalty should not include this overlapped part.

$$\text{mem stall cycle per instruction} = \text{mem miss per instruction} \times (\text{total miss penalty} - \text{overlapped miss penalty})$$

- For the previous example. Suppose 30% of the 75ns miss penalty can be overlapped, what is the AMAT and CPU time?
 - Assume using direct map cache, cc=1.25ns to handle out of order execution.

$$\text{AMATd} = 1*1.25 + 0.014*(75*0.7) = 1.985 \text{ ns}$$

$$\text{CPU time} = (2*1.25 + 1.5 * 0.014 * (75*0.7))*IC = 3.6025 \text{ IC}$$

11/21/2003

Lec. 14

9

Improving CPU Performance

- In the past 10 years, there are over 5000 research papers on reducing the gap between the CPU and memory speeds.
- We will address 17 of them (hopefully) in four categories:
 - Reducing the cache miss penalty
 - Reducing the miss rate
 - Reducing the cache miss penalty or miss rate via parallelism
 - Reducing the hit time

11/21/2003

Lec. 14

10

Cache Misses

- Types of cache misses
 - the three Cs:
 - Compulsory: first access to a block is a miss.
 - Conflict: collision misses, blocks map to the same set.
 - Capacity: replaced blocks that are later referenced, cache too small.
 - the fourth C:
 - Coherence: shared memory, invalid copies
- Relative effects
 - Fully associative placement: no conflict misses.
 - Larger block size *might* reduce compulsory misses.
 - Larger caches have lower capacity misses.

Most of these have negative impacts on hit time and therefore cycle time.

 - a direct mapped cache can be faster than a set associative one

11/21/2003

Lec. 14

11

Reducing Cache Miss Penalty (1)

- Multilevel Caches – “the more the merrier”
 - Add another level behind L1 cache to speed up access from memory (why not combine the two levels into one? Because larger cache will incur longer access time and could stretch cc to hurt all instructions!)

$$\text{AMAT} = \text{Hit time}_{L1} + \text{Miss rate}_{L1} \times (\text{Hit time}_{L2} + \text{Local miss rate}_{L2} \times \text{Miss penalty}_{L2})$$

$$\text{Average memory stall time} = \text{Miss rate}_{L1} \times \text{Hit time}_{L2} + \text{Miss rate}_{L1} \times \text{Local miss rate}_{L2} \times \text{Miss penalty}_{L2}$$

$$\text{Average memory stalls per instruction} = \text{Miss per instruction}_{L1} \times \text{Hit time}_{L2} + \text{Miss per instruction}_{L2} \times \text{Miss penalty}_{L2}$$

11/21/2003

Lec. 14

12

Example

- For every 1000 instructions, 40 misses in L1 and 20 misses in L2; Hit cycle in L1 is 1, L2 is 10; Miss penalty from L2 to memory is 100 cycles; there are 1.5 memory references per instruction. What is AMAT and average stall cycles per instruction?
 - AMAT = $[1 + 40/1000 * (10 + 20/40 * 100)] * cc = 3.4cc$
 - Average stall cycles per instruction = $1.5 * 40/1000 * 10 + 1.5 * 20/1000 * 100 = 3.6$ cycles
- Note: We have not distinguished reads and writes. Access L2 only on L1 miss, i.e. write back cache
- L2 cache design parameters:
 - L1 speed determines the clock rate of CPU, L2 speed affects the miss penalty of L1 cache.
 - $L1 \subset L2$ (multilevel inclusion): L2 size is significantly larger than L1.
 - Block size in L2 > or = block size in L1
 - $L1 \cap L2 = \emptyset$ (multilevel exclusion): L2 size is comparable to L1 (AMD Athlon).

11/21/2003

Lec. 14

13

Reducing Cache Miss Penalty (2,3)

2. Critical word first – “impatience”

L2 cache block:

1	2	3	4	5	6	7	8
---	---	---	---	---	---	---	---

Requested word: 5

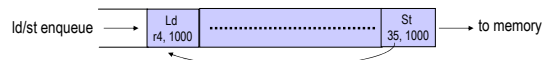
Critical word first:

5	6	7	8	1	2	3	4
---	---	---	---	---	---	---	---

(wrapped fetch)

3. Serves reads before writes have been completed – “preference”

- Recall: In ooo processor, the load/store queue contains loads/stores (waiting for address computation or memory) in program order.



11/21/2003

Lec. 14

14

Reducing Cache Miss Penalty (3,4)

- Complications with write buffer: it stores updated blocks but the L2 hasn't seen them yet! What will happen on a L1 read miss?
 - Conventionally, read miss stalls and waits until write buffer flushes its content to L2 and then access L2 (slow).
 - Alternatively, L1 read miss should check write buffer before going to L2 (faster).
- 4. Improving write buffer (for wrt thru \$) efficiency – “companionship”
 - Two writes with the same address are coalesced.
 - Writes stall if no empty entries are present -> utilize the entries efficiently using *write merge*.

11/21/2003

Lec. 14

15

Write Merge

Write address	V	V	V	V
100	1 Mem[100]	0	0	0
108	1 Mem[108]	0	0	0
116	1 Mem[116]	0	0	0
124	1 Mem[124]	0	0	0

Write address	V	V	V	V
100	1 Mem[100]	1 Mem[108]	1 Mem[116]	1 Mem[124]
	0	0	0	0
	0	0	0	0
	0	0	0	0

11/21/2003

Lec. 14

16

Reducing Cache Miss Penalty (5)

- Victim Cache – “recycling”
 - Holds victim blocks discarded from the L1 cache due to replacement.
 - Small (otherwise an L2) and fully associative.
 - Checked on a L1 miss. If found, block is swapped back to L1 (the block previously took its place is put into victim cache).
 - Works better for small L1 caches since it saves victim blocks from conflict misses.
 - Effective: a 4-entry vc can reduce ¼ of the misses in a 4KB L1 cache [Jouppi, 1990].

11/21/2003

Lec. 14

17