CS510 Computer Architecture

Lecture 14: Cache Write Policies & 2-Level Caches

Soontae Kim

Spring 2017

School of Computing, KAIST

Notice

- **Term project proposal**
  - On May 10 (Wednesday)
  - Prepare less than 10 slides in 5 minutes.
    - **No proposal report required**
  - All team members must prepare parts of presentation.
  - Any topic related to computer architecture; if you are not sure that your topic is appropriate, you can discuss with me.
  - Practice your presentation before coming to class several times so that you can finish your presentation in time; we have only 75 minutes for all of your presentations.
  - You may need to change your topic or direction if your topic conflicts with other team's topic or you find a difficulty in performing your projects.

Cache Write Miss Policy

- **Since data are usually not needed immediately on a write miss, two options exist on a cache write miss:**

Write Allocate:

The cache block is loaded on a write miss followed by write hit actions.

No-Write Allocate:

The block is modified in the lower level (lower cache level, or main memory) and not loaded into cache.

While any of the above two write miss policies can be used with either write back or write through:

- **Write back caches always use write allocate to capture subsequent writes to the block in cache.**
- **Write through caches usually use no-write allocate since subsequent writes still have to go to memory.**

Memory Access Tree, Unified L_1

Write Through, No Write Allocate, No Write Buffer

CPU Memory Access

% reads **100%** **% write**

L_1

Read

% reads x H1

% reads x (1 - H1)

L1 Read Hit:
Hit Access Time = 1
Stalls = 0

L1 Read Miss:
Access Time = M + 1
Stalls Per access = M
Stalls = % reads x (1 - H1) x M

Write

% write x H1

% write x (1 - H1)

L1 Write Hit:
Access Time: M +1
Stalls Per access = M
Stalls =% write x (H1) x M

L1 Write Miss:
Access Time : M + 1
Stalls per access = M
Stalls = % write x (1 - H1) x M

Stall Cycles Per Memory Access = % reads x (1 - H1) x M + % write x M

AMAT = 1 + % reads x (1 - H1) x M + % write x M

CPI = $CPI_{execution}$ + (1 + fraction of loads/stores) x Stall Cycles per access

M = Miss Penalty
H1 = Level 1 Hit Rate
1- H1 = Level 1 Miss Rate

Reducing Write Stalls For Write Through Cache using Write Buffers

- **To reduce write stalls when write through is used, a write buffer is used to eliminate or reduce write stalls:**
  - **Perfect write buffer:** **All writes are handled by write buffer, no stalling for writes**
  - **In this case (for unified L1 cache):**

    Stall Cycles Per Memory Access = % reads x (1 - H1) x M

    (i.e No stalls at all for writes)

  - **Realistic Write buffer:** **Write stalls are not eliminated when the write buffer is full.**
  - **In this case (for unified L1 cache):**

Stall Cycles/Memory Access = (% reads x (1 - H1) + (% write stalls not eliminated) x M

Write Through Cache Performance Example

- **A CPU with $CPI_{execution}$ = 1.1 Mem accesses per instruction = 1.3**
- **Uses a unified L1 <u>Write Through, No Write Allocate</u>, with:**
  - **<u>No write buffer.</u>**
  - **<u>Perfect Write buffer</u>**
  - **<u>A realistic write buffer</u> that eliminates 85% of write stalls**
- **Instruction mix: 50% arith/logic, 15% load, 15% store, 20% control**
- **Assume a cache miss rate of 1.5% and a miss penalty of 50 cycles.**

$$CPI = CPI_{execution} + \text{mem stalls per instruction}$$

% reads = 1.15/1.3 = 88.5% % writes = .15/1.3 = 11.5%

<u>With No Write Buffer</u> :

Mem Stalls/ instruction = 1.3 x 50 x (88.5% x 1.5% + 11.5%) = 8.33 cycles

CPI = 1.1 + 8.33 = 9.43

<u>With Perfect Write Buffer (all write stalls eliminated):</u>

Mem Stalls/ instruction = 1.3 x 50 x (88.5% x 1.5%) = 0.86 cycles

CPI = 1.1 + 0.86 = 1.96

<u>With Realistic Write Buffer (eliminates 85% of write stalls)</u>

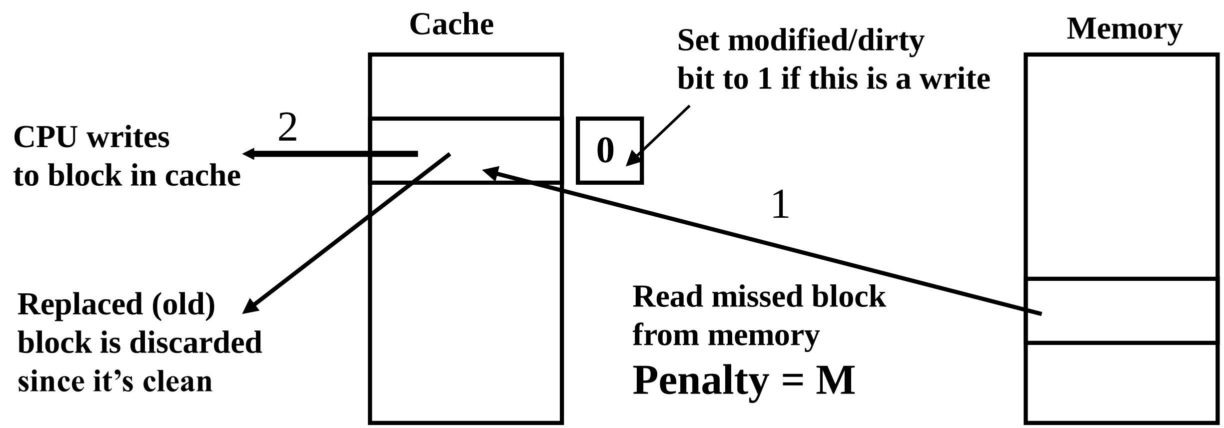
Mem Stalls/ instruction = 1.3 x 50 x (88.5% x 1.5% + 15% x 11.5%) = 1.98 cycles

CPI = 1.1 + 1.98 = 3.08

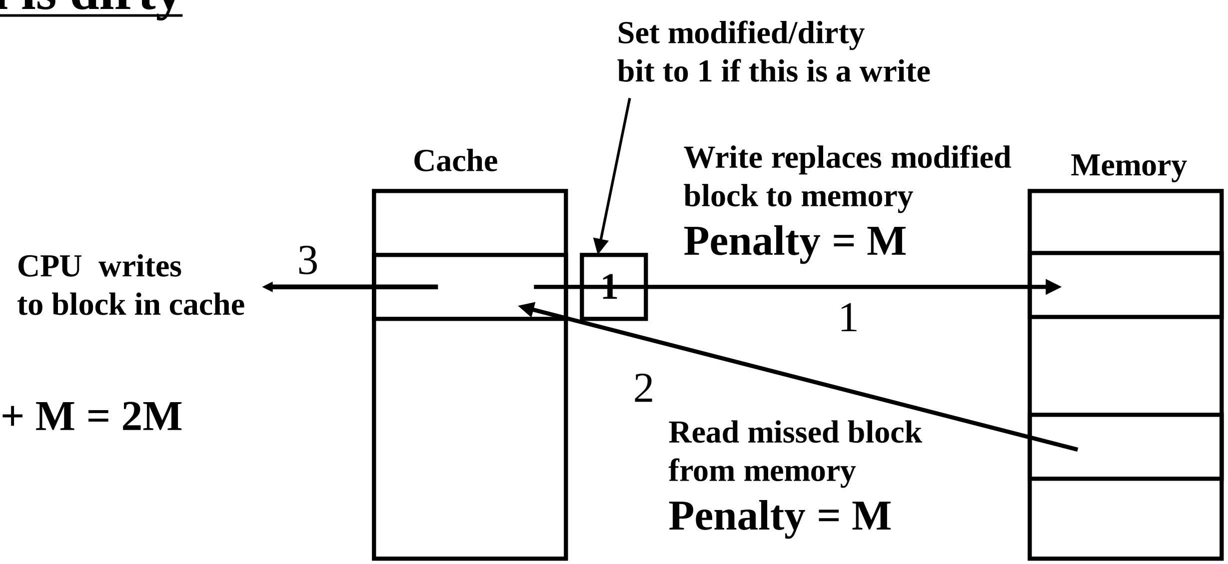
Write Back Cache With Write Allocate: Cache Miss Operation

Block to be replaced is clean

Miss Penalty = M



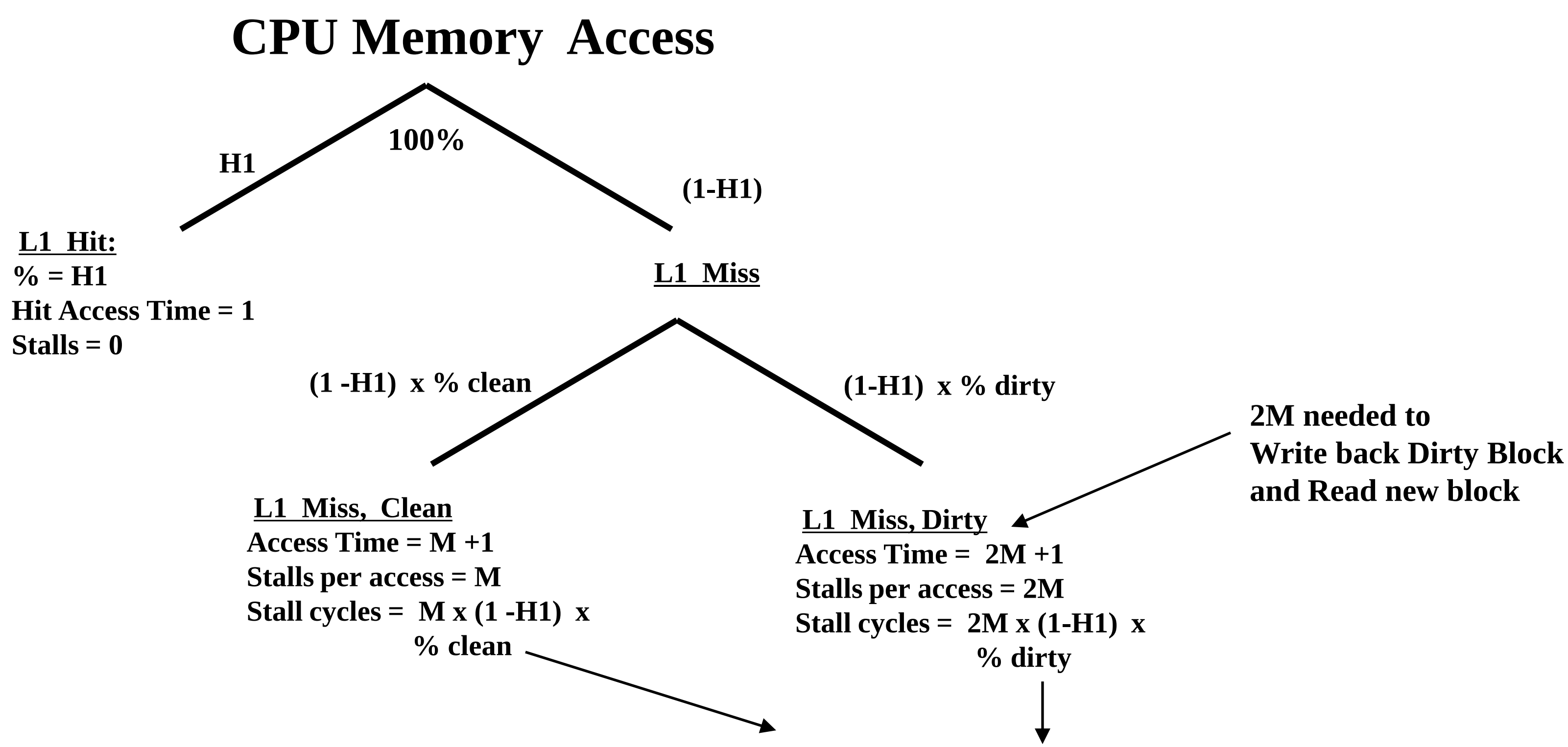
Block to be replaced is dirty



Total Miss Penalty = M + M = 2M

M = Miss Penalty = stall cycles per access resulting from missing in cache

Memory Access Tree Unified L_1 Write Back, With Write Allocate



Stall Cycles Per Memory Access = (1-H1) x (M x % clean + 2M x % dirty)

AMAT = 1 + Stall Cycles Per Memory Access

CPI = $CPI_{execution}$ + (1 + fraction of loads/stores) x Stall Cycles per access

Write Back Cache Performance Example

- **A CPU with $CPI_{execution}$ = 1.1 uses a unified L1 with write back, with write allocate, and the probability a cache block is dirty = 10%**
- **Instruction mix: 50% arith/logic, 15% load, 15% store, 20% control**
- **Assume a cache miss rate of 1.5% and a miss penalty of 50 cycles.**

CPI = $CPI_{execution}$ + mem stalls per instruction

Mem Stalls per instruction =

Mem accesses per instruction x Stalls per access

Mem accesses per instruction = 1 + .3 = 1.3

Stalls per access = (1-H1) x (M x % clean + 2M x % dirty)

Stalls per access = 1.5% x (50 x 90% + 50 x 2 x 10%) = .825 cycles

Mem Stalls per instruction = 1.3 x .825 = 1.07 cycles

AMAT = 1 + 0.825 = 1.825 cycles

CPI = 1.1 + 1.07 = 2.17

The ideal CPU with no misses is 2.17/1.1 = 1.97 times faster

2-Level Cache: L_1, L_2

CPU

L_1 Cache

Hit Rate= H_1,
Hit Access Time = 1 cycle (No Stall)
Stalls for hit access = T_1 = 0

L_2 Cache

Local Hit Rate= H_2
Stalls per hit access= T_2
Hit Access Time = T_2 + 1 cycles

Main Memory

Memory access penalty, M
(stalls per main memory access)
Access Time = M +1

<u>Goal of multi-level Caches:</u>

Reduce the effective miss penalty incurred by level 1 cache misses using additional levels of cache that capture some of these misses

Miss Rates For Multi-Level Caches

- **<u>Local Miss Rate:</u>** **This rate is the number of misses in a cache level divided by the number of memory accesses to this level. Local Hit Rate = 1 - Local Miss Rate**

- **<u>Global Miss Rate:</u>** **The number of misses in a cache level divided by the total number of memory accesses generated by the CPU.**
- **Since level 1 receives all CPU memory accesses, for level 1:**

  Local Miss Rate = Global Miss Rate = 1 - H1
- **For level 2 since it only receives those accesses missed in level 1:**

  Local Miss Rate = Miss rate_{L2}= 1- H2

  Global Miss Rate = Miss rate_{L1} x Miss rate_{L2}

  = (1- H1) x (1 - H2)

2-Level Cache Performance (Ignoring Write Policy)

CPUtime = IC x (CPI$_{execution}$ + Mem Stall cycles per instruction) x C

Mem Stall cycles per instruction = Mem accesses per instruction x Stall cycles per access

- **For a system with 2 levels of unified cache, assuming no penalty when found in L_1 cache:**

Stall cycles per memory access =

[miss rate L_1] x [Hit rate L_2 x Hit time L_2

+ Miss rate L_2 x Memory access penalty)] =

$$(1-H1) \times H2 \times T2 \quad + \quad (1-H1)(1-H2) \times M$$

L1 Miss, L2 Hit

H1 = L1 Hit Rate
H2 = Local L2 Hit Rate
T2 =stall cycles per L2 hit

L1 Miss, L2 Miss:
Must Access Main Memory

2-Level Cache (Both Unified) Performance Memory Access Tree (Ignoring Write Policy)

CPU Stall Cycles Per Memory Access

CPU Memory Access

100%

H1

L_1

<u>L1 Hit:</u>
Hit Access Time = 1
Stalls= H1 x 0 = 0
(No Stall)

L1 Miss:
% = (1-H1)

(1-H1) x H2

(1-H1)(1-H2)

L_2

<u>L1 Miss, L2 Hit:</u>
Hit Access Time =T2 +1
Stalls per L2 Hit = T2
Stalls =(1-H1) x H2 x T2

<u>L1 Miss, L2 Miss:</u>
Access Time = M +1
Stalls per access = M
Stalls= (1-H1)(1-H2) x M

Stall cycles per memory access = (1-H1) x H2 x T2 + (1-H1)(1-H2) x M
AMAT = 1 + (1-H1) x H2 x T2 + (1-H1)(1-H2) x M

Two-Level Cache Example

- **CPU with $CPI_{execution}$ = 1.1 running at clock rate = 500 MHz**
- **1.3 memory accesses per instruction.**
- **L_1 hit access time = 1 cycle (no stall on a hit), a miss rate of 5%**
- **L_2 hit access time = 3 cycles (T2= 2 stall cycles per hit) with local miss rate 40%,**
- **Memory access penalty, M = 100 cycles (stalls per access).** **Find CPI ...**

CPI = $CPI_{execution}$ + Mem Stall cycles per instruction

With No Cache, CPI = 1.1 + 1.3 x 100 = 131.1

With single L_1, CPI = 1.1 + 1.3 x .05 x 100 = 7.6

Stall cycles per memory access = (1-H1) x H2 x T2 + (1-H1)(1-H2) x M

= .05 x .6 x 2 + .05 x .4 x 100

= .06 + 2 = 2.06

AMAT = 2.06 + 1 = 3.06

Mem Stall cycles per instruction = Mem accesses per instruction x Stall cycles per access

= 1.3 x 2.06 = 2.678

CPI = 1.1 + 2.678 = 3.778

Speedup = 7.6/3.778 = 2

Write Policy For 2-Level Cache

- **<u>Write Policy For Level 1 Cache:</u>**
  - **Usually Write through to Level 2**
  - **Write allocate is used to reduce level 1 miss reads.**
  - **Use write buffer to reduce write stalls**

- **<u>Write Policy For Level 2 Cache:</u>**
  - **Usually write back with write allocate is used.**
    - **To minimize memory bandwidth usage.**

- **The above 2-level cache write policy results in <u>inclusive L2 cache</u> since the content of L1 is also in L2**
    - **Common in the majority of CPUs with 2-levels of cache**
    - **As opposed to exclusive L1, L2 (e.g AMD Athlon XP)**

2-Level (Both Unified) Memory Access Tree

L1: Write Through to L2, Write Allocate, With Perfect Write Buffer
L2: Write Back with Write Allocate

CPU Memory Access

(H1) 100% (1-H1) L_1

L1 Hit:
Hit Access Time = 1
Stalls Per access = 0

L1 Miss:

(1-H1) x H2

L_2

L1 Miss, L2 Hit:
Hit Access Time =T2 +1
Stalls per L2 Hit = T2
Stalls = (1-H1) x H2 x T2

(1-H1) x (1-H2)

L1 Miss, L2 Miss

L1 Miss, L2 Miss, Clean
Access Time = M +1
Stalls per access = M
Stall cycles =
M x (1 -H1) x (1-H2) x % clean

L1 Miss, L2 Miss, Dirty
Access Time = 2M +1
Stalls per access = 2M
Stall cycles =
2M x (1-H1) x (1-H2) x % dirty

Stall cycles per memory access = (1-H1) x H2 x T2 + M x (1 -H1) x (1-H2) x % clean + 2M x (1-H1) x (1-H2) x % dirty

= (1-H1) x H2 x T2 + (1 -H1) x (1-H2) x (% clean x M + % dirty x 2M)

Two-Level Cache Example With Write Policy

- **CPU with $CPI_{execution}$ = 1.1 running at clock rate = 500 MHz**
- **1.3 memory accesses per instruction.**
- **For L_1 :**
  - **Cache operates at 500 MHz (no stall on L1 Hit) with a miss rate of 1-H1 = 5%**
  - **Write through to L_2 with perfect write buffer with write allocate**
- **For L_2:**
  - **Hit access time = 3 cycles (T2= 2 stall cycles per hit) local miss rate 1- H2 = 40%**
  - **Write back to main memory with write allocate**
  - **Probability a cache block is dirty = 10%**
- **Memory access penalty, M = 100 cycles. Find CPI.**
- Stall cycles per memory access = (1-H1) x H2 x T2 +

  (1 -H1) x (1-H2) x (% clean x M + % dirty x 2M)

  = .05 x .6 x 2 + .05 x .4 x (.9 x 100 + .1 x200)

  = .06 + 0.02 x 110 = .06 + 2.2 = 2.26
- AMAT = 2.26 + 1 = 3.26 cycles

Mem Stall cycles per instruction = Mem accesses per instruction x Stall cycles per access

= 1.3 x 2.26 = 2.938

CPI = 1.1 + 2.938 = 4.038