The Memory Hierarchy & Cache

- Review of Memory Hierarchy & Cache Basics (from 550):
  - Motivation for The Memory Hierarchy:
    - CPU/Memory Performance Gap
    - The Principle Of Locality
  - Cache Basics:
    - Block placement strategy & Cache Organization:
    - Block replacement policy
    - Unified vs. Separate Level 1 Cache
  - CPU Performance Evaluation with Cache:
    - Average Memory Access Time (AMAT)/Memory Stall cycles
    - Memory Access Tree
  - Classification of Steady-State Cache Misses: The Three C’s of cache Misses
  - Cache Write Policies/Performance Evaluation:
    - Write Though
    - Write Back
  - Cache Write Miss Policies: Cache block allocation policy on a write miss.
  - Multi-Level Caches:
    - Miss Rates For Multi-Level Caches
    - 2-Level Cache Performance
    - Write Policy For 2-Level Cache
    - 3-Level Cache Performance

Cache exploits memory access locality to:
- Lower AMAT by hiding long main memory access latency. Thus cache is considered a memory latency-hiding technique.
- Lower demands on main memory bandwidth.

(Chapter 5.1-5.3)
Memory Hierarchy: Motivation

- The gap between CPU performance and main memory has been widening with higher performance CPUs creating performance bottlenecks for memory access instructions.
- The memory hierarchy is organized into several levels of memory with the smaller, faster memory levels closer to the CPU: registers, then primary Cache Level (L₁), then additional secondary cache levels (L₂, L₃…), then main memory, then mass storage (virtual memory).
- Each level of the hierarchy is usually a subset of the level below: data found in a level is also found in the level below (farther from CPU) but at lower speed (longer access time).
- Each level maps addresses from a larger physical memory to a smaller level of physical memory closer to the CPU.
- This concept is greatly aided by the principal of locality both temporal and spatial which indicates that programs tend to reuse data and instructions that they have used recently or those stored in their vicinity leading to working set of a program.

For Ideal Memory: Memory Access Time = 1 CPU cycle

(Review from 550) (Chapter 5.1-5.3)
Memory Hierarchy: Motivation
Processor-Memory (DRAM) Performance Gap

(i.e. Gap between memory access time and CPU cycle time)

μProc 60%/yr.

Processor-Memory Performance Gap:
(grows 50% / year)

Ideal Memory Access Time = 1 CPU Cycle
Real Memory Access Time >> 1 CPU cycle
### Processor-DRAM Performance Gap Impact

- To illustrate the performance impact, assume a single-issue pipelined CPU with CPI = 1 using non-ideal memory.
- Ignoring other factors, the minimum cost of a full memory access in terms of number of wasted CPU cycles:

<table>
<thead>
<tr>
<th>Year</th>
<th>CPU speed MHZ</th>
<th>CPU cycle ns</th>
<th>Memory Access ns</th>
<th>Minimum CPU memory stall cycles or instructions wasted</th>
</tr>
</thead>
<tbody>
<tr>
<td>1986:</td>
<td>8</td>
<td>125</td>
<td>190</td>
<td>190/125 - 1 = 0.5</td>
</tr>
<tr>
<td>1989:</td>
<td>33</td>
<td>30</td>
<td>165</td>
<td>165/30 -1 = 4.5</td>
</tr>
<tr>
<td>1992:</td>
<td>60</td>
<td>16.6</td>
<td>120</td>
<td>120/16.6 -1 = 6.2</td>
</tr>
<tr>
<td>1996:</td>
<td>200</td>
<td>5</td>
<td>110</td>
<td>110/5 -1 = 21</td>
</tr>
<tr>
<td>1998:</td>
<td>300</td>
<td>3.33</td>
<td>100</td>
<td>100/3.33 -1 = 29</td>
</tr>
<tr>
<td>2000:</td>
<td>1000</td>
<td>1</td>
<td>90</td>
<td>90/1 - 1 = 89</td>
</tr>
<tr>
<td>2002:</td>
<td>2000</td>
<td>.5</td>
<td>80</td>
<td>80/.5 - 1 = 159</td>
</tr>
<tr>
<td>2004:</td>
<td>3000</td>
<td>.333</td>
<td>60</td>
<td>60.333 - 1 = 179</td>
</tr>
</tbody>
</table>

(Review from 550)

Ideal Memory Access Time = 1 CPU Cycle
Real Memory Access Time >> 1 CPU cycle
Memory Hierarchy: Motivation

The Principle Of Locality

• Programs usually access a relatively small portion of their address space (instructions/data) at any instant of time (program working set).

  Thus: Memory Access Locality $\rightarrow$ Program Working Set

• Two Types of access locality:
  – **Temporal Locality**: If an item (instruction or data) is referenced, it will tend to be referenced again soon.
    • e.g. instructions in the body of inner loops
  – **Spatial locality**: If an item is referenced, items whose addresses are close will tend to be referenced soon.
    • e.g. sequential instruction execution, sequential access to elements of array

• The presence of locality in program behavior (memory access patterns), makes it possible to satisfy a large percentage of program memory access needs (both instructions and data) using faster memory levels (cache) with much less capacity than program address space.

(Review from 550)
Access Locality & Program Working Set

• Programs usually access a relatively small portion of their address space (instructions/data) at any instant of time (program working set).

• The presence of locality in program behavior and memory access patterns, makes it possible to satisfy a large percentage of program memory access needs using faster memory levels with much less capacity than program address space.

Program Instruction Address Space

Program Data Address Space

Locality in program memory access → Program Working Set
Levels of The Memory Hierarchy

Part of The On-chip
CPU Datapath
ISA 16-128 Registers

One or more levels (Static RAM):
Level 1: On-chip 16-64K
Level 2: On-chip 256K-2M
Level 3: On or Off-chip 1M-32M

Dynamic RAM (DRAM)
256M-16G

Cache Level(s)

Main Memory

Magnetic Disc

Optical Disk or Magnetic Tape

Farther away from
the CPU:
Lower Cost/Bit
Higher Capacity
Increased Access
Time/Latency
Lower Throughput/
Bandwidth

(Virtual Memory)

Faster Access
Time

(Review from 550)
Memory Hierarchy Operation

• If an instruction or operand is required by the CPU, the levels of the memory hierarchy are searched for the item starting with the level closest to the CPU (Level 1 cache):
  – If the item is found, it’s delivered to the CPU resulting in a cache hit without searching lower levels.
  – If the item is missing from an upper level, resulting in a cache miss, the level just below is searched.
  – For systems with several levels of cache, the search continues with cache level 2, 3 etc.
  – If all levels of cache report a miss then main memory is accessed for the item.

• CPU ↔ cache ↔ memory: Managed by hardware.

• Memory ↔ disk: Managed by the operating system with hardware support.

(Review from 550)
Memory Hierarchy: Terminology

- **A Block**: The smallest unit of information transferred between two levels.
- **Hit**: Item is found in some block in the upper level (example: Block X)
  - **Hit Rate**: The fraction of memory access found in the upper level.
  - **Hit Time**: Time to access the upper level which consists of
    RAM access time + Time to determine hit/miss
- **Miss**: Item needs to be retrieved from a block in the lower level (Block Y)
  - **Miss Rate** = 1 - (Hit Rate)
  - **Miss Penalty**: Time to replace a block in the upper level + M
    Time to deliver the missed block to the processor
- **Hit Time << Miss Penalty M**
Basic Cache Concepts

- Cache is the first level of the memory hierarchy once the address leaves the CPU and is searched first for the requested data.

- If the data requested by the CPU is present in the cache, it is retrieved from cache and the data access is a **cache hit** otherwise a **cache miss** and data must be read from main memory.

- On a cache miss a block of data must be brought in from main memory to cache to possibly **replace** an existing cache block.

- The allowed block addresses where blocks can be mapped (placed) into cache from main memory is determined by **cache placement strategy**.

- Locating a block of data in cache is handled by cache **block identification mechanism**: Tag matching.

- On a cache miss choosing the cache block being removed (replaced) is handled by the **block replacement strategy** in place.

- When a write to cache is requested, a number of main memory update strategies exist as part of **the cache write policy**.

(Review from 550)
Basic Cache Design & Operation Issues

• Q1: Where can a block be placed cache?  
  (Block placement strategy & Cache organization)  
  – Fully Associative, Set Associative, Direct Mapped.

• Q2: How is a block found if it is in cache?  
  (Block identification)  
  – Tag/Block.

• Q3: Which block should be replaced on a miss?  
  (Block replacement)  
  – Random, LRU, FIFO.

• Q4: What happens on a write?  
  (Cache write policy)  
  – Write through, write back.
Cache Organization & Placement Strategies

Placement strategies or mapping of a main memory data block onto cache block frames divide cache designs into three organizations:

1. **Direct mapped cache:** A block can be placed in only one location (cache block frame), given by the mapping function:
   \[
   \text{index} = \frac{\text{Block address}}{\text{Number of blocks in cache}}
   \]

   Least complex to implement

2. **Fully associative cache:** A block can be placed anywhere in cache. (no mapping function).

   Most complex cache organization to implement

3. **Set associative cache:** A block can be placed in a restricted set of places, or cache block frames. A set is a group of block frames in the cache. A block is first mapped onto the set and then it can be placed anywhere within the set. The set in this case is chosen by:
   \[
   \text{index} = \frac{\text{Block address}}{\text{Number of sets in cache}}
   \]

   If there are \( n \) blocks in a set the cache placement is called \( n \)-way set-associative.

(Review from 550)
Cache Organization: Direct Mapped Cache

A block can be placed in one location only, given by:
(Block address) MOD (Number of blocks in cache)

In this case, mapping function: (Block address) MOD (8)

Index bits

Example:
29 MOD 8 = 5
(11101) MOD (1000) = 101

Limitation of Direct Mapped Cache: Conflicts between memory blocks that map to the same cache block frame
4KB Direct Mapped Cache Example

1K = 1024 Blocks
Each block = one word

Can cache up to
$2^{32}$ bytes = 4 GB of memory

Mapping function:

Cache Block frame number = (Block address) MOD (1024)
i.e. index field or 10 low bit of block address

(Review from 550)

Direct mapped cache is the least complex cache organization

#14 lec #8 Winter 2005 1-23-2006
64KB Direct Mapped Cache Example

4K = 4096 blocks
Each block = four words = 16 bytes
Can cache up to $2^{32}$ bytes = 4 GB of memory

Larger cache blocks take better advantage of spatial locality and thus may result in a lower miss rate

Mapping Function: Cache Block frame number = (Block address) MOD (4096)
i.e. index field or 12 low bit of block address

(Review from 550)
Cache Organization:
Set Associative Cache

Set associative cache reduces cache misses by reducing conflicts between blocks that would have been mapped to the same cache block frame in the case of direct mapped cache.

1-way set associative: (direct mapped) 1 block frame per set

2-way set associative: 2 blocks frames per set

4-way set associative: 4 blocks frames per set

8-way set associative: 8 blocks frames per set
In this case it becomes fully associative since total number of block frames = 8

(Review from 550)
Cache Organization/Mapping Example

- **Fully associative:** block 12 can go anywhere (No mapping function)
- **Direct mapped:** block 12 can go only into block 4 (12 mod 8)
- **2-way:** Set associative: block 12 can go anywhere in set 0 (12 mod 4)

8 Block Frames:

32 Block Frames:

This example cache has eight block frames and memory has 32 blocks.

(Review from 550)
4K Four-Way Set Associative Cache: MIPS Implementation Example

1024 block frames
Each block = one word
4-way set associative
1024 / 4 = 256 sets

Can cache up to
2^{32} bytes = 4 GB
of memory

Set associative cache requires parallel tag matching and more complex hit logic which may increase hit time

Block Address = 30 bits
Tag = 22 bits
Index = 8 bits
Offset = 2 bits

Mapping Function: Cache Set Number = index = (Block address) MOD (256)
Locating A Data Block in Cache

- Each block frame in cache has an address tag.
- The tags of every cache block that might contain the required data are checked in parallel.
- A valid bit is added to the tag to indicate whether this entry contains a valid address.
- The address from the CPU to cache is divided into:
  - A block address, further divided into:
    - An index field to choose a block frame/set in cache.
      (no index field when fully associative).
    - A tag field to search and match addresses in the selected set.
  - A block offset to select the data from the block.
Address Field Sizes/Mapping

Physical Memory Address Generated by CPU
(size determined by amount of physical main memory cacheable)

Block Address

Tag

Index

Block Offset

Block offset size = \( \log_2(\text{block size}) \)

Index size = \( \log_2(\text{Total number of blocks/associativity}) \)

Tag size = address size - index size - offset size

Mapping function:

Cache set or block frame number = Index =

= (Block Address) MOD (Number of Sets)

Number of Sets in cache

No index/mapping function for fully associative cache

(Review from 550)
Cache Replacement Policy

- When a cache miss occurs the cache controller may have to select a block of cache data to be removed from a cache block frame and replaced with the requested data, such a block is selected by one of three methods:

(No cache replacement policy in direct mapped cache)

- **Random:**
  - Any block is randomly selected for replacement providing uniform allocation.
  - Simple to build in hardware. Most widely used cache replacement strategy.

- **Least-recently used (LRU):**
  - Accesses to blocks are recorded and and the block replaced is the one that was not used for the longest period of time.
  - Full LRU is *expensive* to implement, as the number of blocks to be tracked increases, and is usually approximated by block usage bits that are cleared at regular time intervals.

- **First In, First Out (FIFO):**
  - Because LRU can be complicated to implement, this approximates LRU by determining the oldest block rather than LRU

(Review from 550)
Miss Rates for Caches with Different Size, Associativity & Replacement Algorithm

Sample Data

<table>
<thead>
<tr>
<th>Associativity:</th>
<th>2-way</th>
<th>4-way</th>
<th>8-way</th>
</tr>
</thead>
<tbody>
<tr>
<td>Size</td>
<td>LRU</td>
<td>Random</td>
<td>LRU</td>
</tr>
<tr>
<td>16 KB</td>
<td>5.18%</td>
<td>5.69%</td>
<td>4.67%</td>
</tr>
<tr>
<td>64 KB</td>
<td>1.88%</td>
<td>2.01%</td>
<td>1.54%</td>
</tr>
<tr>
<td>256 KB</td>
<td>1.15%</td>
<td>1.17%</td>
<td>1.13%</td>
</tr>
</tbody>
</table>

Program steady state cache miss rates are given
Initially cache is empty and miss rates ~ 100%

FIFO replacement miss rates (not shown here) is better than random but worse than LRU

For SPEC92

(Review from 550)
Unified vs. Separate Level 1 Cache

- **Unified Level 1 Cache (Princeton Memory Architecture).**
  A single level 1 (L₁) cache is used for both instructions and data.

- **Separate instruction/data Level 1 caches (Harvard Memory Architecture):**
  The level 1 (L₁) cache is split into two caches, one for instructions (instruction cache, L₁ I-cache) and the other for data (data cache, L₁ D-cache).

(Review from 550)
Memory Hierarchy Performance:
Average Memory Access Time (AMAT), Memory Stall cycles

- **The Average Memory Access Time (AMAT):** The number of cycles required to complete an average memory access request by the CPU.
- **Memory stall cycles per memory access:** The number of stall cycles added to CPU execution cycles for one memory access.
- **Memory stall cycles per average memory access** = \( (\text{AMAT} - 1) \)
- For ideal memory: AMAT = 1 cycle, this results in zero memory stall cycles.
- **Memory stall cycles per average instruction** =
  \[
  \frac{\text{Number of memory accesses per instruction}}{1 + \text{fraction of loads/stores}} \times (\text{AMAT} - 1)
  \]

Base CPI = \( \text{CPI}_{\text{execution}} = \text{CPI with ideal memory} \)

\[
\text{CPI} = \text{CPI}_{\text{execution}} + \text{Mem Stall cycles per instruction}
\]

(Review from 550)
Cache Performance: Single Level L1 Princeton (Unified) Memory Architecture

CPU time = Instruction count x CPI x Clock cycle time

CPI_{execution} = CPI with ideal memory

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

Mem Stall cycles per instruction =

Memory accesses per instruction x Memory stall cycles per access

Assuming no stall cycles on a cache hit (cache access time = 1 cycle, stall = 0)

Cache Hit Rate = H1  Miss Rate = 1- H1

Memory stall cycles per memory access = Miss rate x Miss penalty = (1 - H1) x M

AMAT = 1 + Miss rate x Miss penalty
Memory accesses per instruction = (1 + fraction of loads/stores)

Miss Penalty = M = the number of stall cycles resulting from missing in cache

= Main memory access time - 1

Thus for a unified L1 cache with no stalls on a cache hit:

CPI = CPI_{execution} + (1 + fraction of loads/stores) x (1 - H1) x M

AMAT = 1 + (1 - H1) x M
**Memory Access Tree:**

For Unified Level 1 Cache

---

**Probability to be here**

**CPU Memory Access**

- **L1 Hit:**
  - % = Hit Rate = H1
  - Hit Access Time = 1
  - Stall cycles per access = 0
  - Stall = H1 x 0 = 0
  - (No Stall)

- **L1 Miss:**
  - % = (1- Hit rate) = (1-H1)
  - Access time = M + 1
  - Stall cycles per access = M
  - Stall = M x (1-H1)

**AMAT** = H1 x 1 + (1 - H1) x (M+ 1) = 1 + M x (1 - H1)

**Stall Cycles Per Access** = AMAT - 1 = M x (1 - H1)

**CPI** = \( \text{CPI}_{\text{execution}} + (1 + \text{fraction of loads/stores}) \times M \times (1 - H1) \)

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

**M + 1** = Miss Time = Main memory access time

**H1** = Level 1 Hit Rate \( 1 - H1 \) = Level 1 Miss Rate

---

(Review from 550)
Cache Performance Example

- Suppose a CPU executes at Clock Rate = 200 MHz (5 ns per cycle) with a single level of cache.
- \( \text{CPI}_{\text{execution}} = 1.1 \)
- Instruction mix: 50% arith/logic, 30% load/store, 20% control
- Assume a cache miss rate of 1.5% and a miss penalty of \( M = 50 \) cycles.

\[
\text{CPI} = \text{CPI}_{\text{execution}} + \text{mem stalls per instruction} \]

\[
\text{Mem Stalls per instruction} = \left(1 - H_1\right) M \]

\[
\text{Mem accesses per instruction} = 1 + 0.3 = 1.3
\]

\[
\text{Mem Stalls per memory access} = \left(1 - H_1\right) M = 0.015 \times 50 = 0.75 \text{ cycles}
\]

\[
\text{AMAT} = 1 + 0.75 = 1.75 \text{ cycles}
\]

\[
\text{Mem Stalls per instruction} = 1.3 \times 0.015 \times 50 = 0.975
\]

\[
\text{CPI} = 1.1 + 0.975 = 2.075
\]

The ideal memory CPU with no misses is \( 2.075/1.1 = 1.88 \) times faster

\( M = \text{Miss Penalty} = \text{stall cycles per access resulting from missing in cache} \)
Cache Performance Example

• Suppose for the previous example we double the clock rate to 400 MHz, how much faster is this machine, assuming similar miss rate, instruction mix?

• Since memory speed is not changed, the miss penalty takes more CPU cycles:

  Miss penalty = M = 50 x 2 = 100 cycles.

  CPI = 1.1 + 1.3 x .015 x 100 = 1.1 + 1.95 = 3.05

  Speedup = \( \frac{(CPI_{\text{old}} \times C_{\text{old}})}{(CPI_{\text{new}} \times C_{\text{new}})} \)

  = \( \frac{2.075 \times 2}{3.05} \) = 1.36

The new machine is only 1.36 times faster rather than 2 times faster due to the increased effect of cache misses.

\(\rightarrow \) CPUs with higher clock rate, have more cycles per cache miss and more memory impact on CPI.
Cache Performance

Harvard Memory Architecture

For a CPU with separate or split level one (L1) caches for instructions and data (Harvard memory architecture) and no stalls for cache hits:

\[ \text{CPU time} = \text{Instruction count} \times \text{CPI} \times \text{Clock cycle time} \]

\[ \text{CPI} = \text{CPI}_{\text{execution}} + \text{Mem Stall cycles per instruction} \]

\[ \text{Mem Stall cycles per instruction} = \]
\[ \text{Instruction Fetch Miss rate} \times M + \]
\[ \text{Data Memory Accesses Per Instruction} \times \text{Data Miss Rate} \times M \]

\[ M = \text{Miss Penalty} = \text{stall cycles per access to main memory resulting from missing in cache} \]
Memory Access Tree
For Separate Level 1 Caches

CPU Memory Access

L1

% Instructions

% data

Instruction

% instructions x Instruction H1

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

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

Data

% data x Data H1

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

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

Ideal access on a hit

Stall Cycles Per Access = % Instructions x (1 - Instruction H1) x M + % data x (1 - Data H1) x M
AMAT = 1 + Stall Cycles per access
Stall cycles per instruction = (1 + fraction of loads/stores) x Stall Cycles per access
CPI = CPI_{execution} + Stall cycles per instruction
= CPI_{execution} + (1 + fraction of loads/stores) x Stall Cycles per access

M = Miss Penalty = stall cycles per access resulting from missing in cache
M + 1 = Miss Time = Main memory access time
Data H1 = Level 1 Data Hit Rate
1- Data H1 = Level 1 Data Miss Rate
Instruction H1 = Level 1 Instruction Hit Rate
1- Instruction H1 = Level 1 Instruction Miss Rate
% Instructions = Percentage or fraction of instruction fetches out of all memory accesses
% Data = Percentage or fraction of data accesses out of all memory accesses

(Review from 550)
Cache Performance Example

- Suppose a CPU uses separate level one (L1) caches for instructions and data (Harvard memory architecture) with different miss rates for instruction and data access:
  - A cache hit incurs no stall cycles while a cache miss incurs 200 stall cycles for both memory reads and writes.
  - CPI_{execution} = 1.1
  - Instruction mix: 50% arith/logic, 30% load/store, 20% control
  - Assume a cache miss rate of 0.5% for instruction fetch and a cache data miss rate of 6%.
  - A cache hit incurs no stall cycles while a cache miss incurs 200 stall cycles for both memory reads and writes. Find the resulting CPI using this cache? How much faster is the CPU with ideal memory?

\[
\text{CPI} = \text{CPI}_{\text{execution}} + \text{mem stalls per instruction}
\]

Mem Stall cycles per instruction = Instruction Fetch Miss rate x Miss Penalty + Data Memory Accesses Per Instruction x Data Miss Rate x Miss Penalty

Mem Stall cycles per instruction = \(0.5/100 \times 200 + 0.3 \times 6/100 \times 200 = 1 + 3.6 = 4.6\)

Mem Stall cycles per access = \(4.6 / 1.3 = 3.5\) cycles AMAT = 1 + 3.5 = 4.5 cycles

\[
\text{CPI} = \text{CPI}_{\text{execution}} + \text{mem stalls per instruction} = 1.1 + 4.6 = 5.7
\]

The CPU with ideal cache (no misses) is 5.7/1.1 = 5.18 times faster

With no cache the CPI would have been = 1.1 + 1.3 \times 200 = 261.1 !!

(Review from 550)
Memory Access Tree For Separate Level 1 Caches Example

(Ignoring Write Policy)

30% of all instructions executed are loads/stores, thus:
Fraction of instruction fetches out of all memory accesses = 1/ (1+0.3) = 1/1.3 = 0.769 or 76.9 %
Fraction of data accesses out of all memory accesses = 0.3/ (1+0.3) = 0.3/1.3 = 0.231 or 23.1 %

CPU Memory Access

L1

Instruction

% instructions x Instruction H1)
= .765 or 76.5 %

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

Ideal access on a hit

Instruction L1 Miss:
Access Time = M + 1 = 201
Stalls Per access = M = 200
Stalls = %instructions x (1 - Instruction H1 ) x M
= 0.003846 x 200 = 0.7692 cycles

% Instructions = 0.769 or 76.9 %

Data

% data = 0.231 or 23.1 %

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

Ideal access on a hit

Data L1 Miss:
Access Time = M + 1 = 201
Stalls per access: = M = 200
Stalls = % data x (1 - Data H1 ) x M
= 0.01385 x 200 = 2.769 cycles

% data x Data H1
= .2169 or 21.69 %

Stall Cycles Per Access = % Instructions x (1 - Instruction H1 ) x M + % data x (1 - Data H1 ) x M
= 0.7692 + 2.769 = 3.54 cycles

AMAT = 1 + Stall Cycles per access = 1 + 3.5 = 4.54 cycles

Stall cycles per instruction = (1 + fraction of loads/stores) x Stall Cycles per access = 1.3 x 3.54 = 4.6 cycles

CPI = CPI_{execution} + Stall cycles per instruction = 1.1 + 4.6 = 5.7

(Review from 550)

M = Miss Penalty = stall cycles per access resulting from missing in cache = 200 cycles
M + 1 = Miss Time = Main memory access time = 200+1 =201 cycles
L1 access Time = 1 cycle
Data H1 = 0.94 or 94% 1- Data H1 = 0.06 or 6% 
Instruction H1 = 0.995 or 99.5% 1- Instruction H1 = 0.005 or 0.5 % 
% Instructions = Percentage or fraction of instruction fetches out of all memory accesses = 76.9 %
% Data = Percentage or fraction of data accesses out of all memory accesses = 23.1 %
## Typical Cache Performance Data Using SPEC92

<table>
<thead>
<tr>
<th>Size</th>
<th>Instruction cache</th>
<th>Data cache</th>
<th>Unified cache</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 KB</td>
<td>3.06%</td>
<td>24.61%</td>
<td>13.34%</td>
</tr>
<tr>
<td>2 KB</td>
<td>2.26%</td>
<td>20.57%</td>
<td>9.78%</td>
</tr>
<tr>
<td>4 KB</td>
<td>1.78%</td>
<td>15.94%</td>
<td>7.24%</td>
</tr>
<tr>
<td>8 KB</td>
<td>1.10%</td>
<td>10.19%</td>
<td>4.57%</td>
</tr>
<tr>
<td>16 KB</td>
<td>0.64%</td>
<td>6.47%</td>
<td>2.87%</td>
</tr>
<tr>
<td>32 KB</td>
<td>0.39%</td>
<td>4.82%</td>
<td>1.99%</td>
</tr>
<tr>
<td>64 KB</td>
<td>0.15%</td>
<td>3.77%</td>
<td>1.35%</td>
</tr>
<tr>
<td>128 KB</td>
<td>0.02%</td>
<td>2.88%</td>
<td>0.95%</td>
</tr>
</tbody>
</table>

Miss rates for instruction, data, and unified caches of different sizes.

Program steady state cache miss rates are given
Initially cache is empty and miss rates ~ 100%
Types of Cache Misses: *The Three C’s*

1. **Compulsory:** On the first access to a block; the block must be brought into the cache; also called cold start misses, or first reference misses.
   - Initially upon program startup: Miss rate ~ 100% All compulsory misses

2. **Capacity:** Occur because blocks are being discarded from cache because cache cannot contain all blocks needed for program execution (program working set is much larger than cache capacity).

3. **Conflict:** In the case of set associative or direct mapped block placement strategies, conflict misses occur when several blocks are mapped to the same set or block frame; also called collision misses or interference misses.
The 3 Cs of Cache:
Absolute Steady State Miss Rates (SPEC92)

(For Unified L1 Cache)
The 3 Cs of Cache:

Relative Steady State Miss Rates (SPEC92)

- **Miss Rate per Type**
  - 1-way
  - 2-way
  - 4-way
  - 8-way

- **Cache Size (KB)**
  - 1
  - 2
  - 4
  - 8
  - 16
  - 32
  - 64
  - 128

- **Total Normalized to 1 or 100%**

- **Compulsory**

---

**Legend**
- 1-way
- 2-way
- 4-way
- 8-way
- Conflict
- Capacity
Cache Read/Write Operations

- Statistical data suggest that reads (*including instruction fetches*) dominate processor cache accesses (writes account for ~ 25% of data cache traffic).

- In cache reads, a block is read at the same time while the tag is being compared with the block address. If the read is a hit the data is passed to the CPU, if a miss it ignores it.

- In cache writes, modifying the block cannot begin until the tag is checked to see if the address is a hit. Thus for cache writes, *tag checking* cannot take place in parallel, and only the specific data (between 1 and 8 bytes) requested by the CPU can be modified.
  
  - *Solution:* Pipeline tag checking and cache write.

- Cache can be classified according to the write and memory update strategy in place as: *write through*, or *write back* cache.

Pipelining of Tag Checking and Cache Write

<table>
<thead>
<tr>
<th>Tag Check</th>
<th>Cache Write</th>
</tr>
</thead>
<tbody>
<tr>
<td>Tag Check</td>
<td>Cache Write</td>
</tr>
</tbody>
</table>

\[\text{i.e. stores} \]

\[\text{i.e. write hit (we have old block to modify in cache)} \]
Cache Write Strategies

1. Write Though: Data is written to both the cache block and to a block of main memory.
   - The lower level always has the most updated data; an important feature for I/O and multiprocessing.
   - Easier to implement than write back.
   - A write buffer is often used to reduce CPU write stall while data is written to memory.

2. Write Back: Data is written or updated only to the cache block. The modified or dirty cache block is written to main memory when it’s being replaced from cache.
   - Writes occur at the speed of cache
   - A status bit called a dirty or modified bit, is used to indicate whether the block was modified while in cache; if not the block is not written back to main memory when replaced.
   - Advantage: Uses less memory bandwidth than write through.
Cache Write Strategies:

**Write Hit Operation** (block to be written to is in cache)

### Write Through

- **Write (Store)**
  - Write to cache
  - **Without Write Buffer:**
    - Write to cache and also to memory
    - Write Penalty = M
  - **With perfect write buffer:**
    - Write to cache and also to write buffer
    - No penalty (no stall)

### For cache write miss:
- With no write allocate
- Similar but no write to cache
- Penalty is still M

**Without Write Buffer:**
- Write Hit = We have old block to be modified in cache

**Write Back**

- **Write (Store)**
  - Just write to cache block and set dirty or modified bit to 1
  - No penalty (no stall)

**Without Write Buffer:**
- Write Back to memory when replaced in cache

**Set modified/dirty bit to 1 to indicate that cache block has been modified (i.e. block is dirty)**
Cache Write Miss Policy

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

**Write Allocate:**
The missed cache block is loaded into cache on a write miss followed by write hit actions. i.e. A cache block frame is allocated for the block to be modified (written-to)

**No-Write Allocate:** i.e. A cache block frame is not allocated for the block to be modified (written-to)
The block is modified in the lower level (lower cache level, or main memory) and not loaded (written or updated) 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.
Write Back Cache With Write Allocate: Cache Miss Operation (read or write miss)

**Block to be replaced is clean**

- CPU reads or writes to block in cache
- Block to be replaced is clean
- Set modified/dirty bit to 1 if this is a write
- Read missed block from memory
- Penalty = M
- Miss Penalty = M

**Block to be replaced is dirty (modified)**

1. Write replaced modified block to memory
   - Penalty = M
2. Read missed block from memory
   - Penalty = M

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 Through, No Write Allocate, No Write Buffer

**CPU Memory Access**

- **Read**
  - $L_1$ Read Hit: Hit Access Time = 1
  - Stalls = 0

  - $L_1$ Read Miss:
    - Access Time = $M + 1$
    - Stalls Per access = $M$
    - Stalls = % reads x $(1 - H_1) x M$

- **Write**
  - $L_1$ Write Hit:
    - Access Time: $M + 1$
    - Stalls Per access = $M$
    - Stalls = % write x $H_1 x M$

  - $L_1$ Write Miss:
    - Access Time: $M + 1$
    - Stalls per access = $M$
    - Stalls = % write x $(1 - H_1) x M$

*Stall Cycles Per Memory Access = % reads x $(1 - H_1) x M + % write x M*  

*AMAT = 1 + % reads x $(1 - H_1) x M + % write x M*  

*M = Miss Penalty*  

*M + 1 = Miss Time = Main memory access time*  

*H_1 = Level 1 Hit Rate*  

*1 - H_1 = Level 1 Miss Rate*  

*Stall Cycles per access = AMAT - 1*  

*Stall Cycles per access = CPIexecution + (1 + fraction of loads/stores) x Stall Cycles per access*  

*AMAT = 1 + % reads x $(1 - H_1) x M + % write x M*  

*AMAT = CPIexecution + (1 + fraction of loads/stores) x Stall Cycles per access*  

*Stall Cycles per access = AMAT - 1*
Reducing Write Stalls For Write Though Cache

Write Buffers

- To reduce write stalls when write though 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):
    
    \[ \text{Stall Cycles Per Memory Access} = \% \text{ reads } \times (1 - H_1) \times M \]
    
    (i.e No stalls at all for writes)

  - **Realistic Write buffer:** A percentage of write stalls are not eliminated when the write buffer is full.
    
  - **In this case** (for unified L1 cache):
    
    \[ \text{Stall Cycles/Memory Access} = (\% \text{ reads } \times (1 - H_1) + \% \text{ write stalls not eliminated}) \times M \]
Write Through Cache Performance Example

- A CPU with $CPI_{\text{execution}} = 1.1$ Mem accesses per instruction = 1.3
- Uses a unified L1 Write Through, No Write Allocate, with:
  - No write buffer.
  - Perfect Write buffer
  - A realistic write buffer 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.

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

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

With No Write Buffer:

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

$CPI = 1.1 + 8.33 = 9.43$

With Perfect Write Buffer (all write stalls eliminated):

Mem Stalls/ instruction = $1.3 \times 50 \times (88.5\% \times 1.5\%) = 0.86$ cycles

$CPI = 1.1 + 0.86 = 1.96$

With Realistic Write Buffer (eliminates 85% of write stalls)

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

$CPI = 1.1 + 1.98 = 3.08$
Memory Access Tree Unified L₁
Write Back, With Write Allocate

CPU Memory Access

L₁ Hit:
% = H₁
Hit Access Time = 1
Stalls = 0

L₁ Hit:
% = H₁
Hit Access Time = 1
Stalls = 0

L₁ Miss

(1-H₁)

L₁ Miss

(1-H₁)

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

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

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

AMAT = 1 + Stall Cycles Per Memory Access

CPI = CPI_{\text{execution}} + (1 + \text{fraction of loads/stores}) \times \text{Stall Cycles per access}

M = Miss Penalty = stall cycles per access resulting from missing in cache
M + 1 = Miss Time = Main memory access time
H₁ = Level 1 Hit Rate
1 - H₁ = Level 1 Miss Rate

EECC551 - Shaaban

#45 lec # 8 Winter 2005 1-23-2006
Write Back Cache Performance Example

- A CPU with $CPI_{\text{execution}} = 1.1$ uses a unified L1 with write back, 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.

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

Mem Stalls per instruction =

$$\text{Mem accesses per instruction} \times \text{Stalls per access}$$

Mem accesses per instruction $= 1 + 0.3 = 1.3$

Stalls per access $= \frac{1}{1 - H1} \times (M \times % \text{clean} + 2M \times % \text{dirty})$

Stalls per access $= 1.5\% \times (50 \times 90\% + 100 \times 10\%) = 0.825$ cycles

AMAT $= 1 + \text{stalls per access} = 1 + 0.825 = 1.825$ cycles

Mem Stalls per instruction $= 1.3 \times 0.825 = 1.07$ cycles

$$CPI = 1.1 + 1.07 = 2.17$$

The ideal CPU with no misses is $2.17/1.1 = 1.97$ times faster
Memory Access Tree For Unified L₁
Write Back, With Write Allocate Example

**CPU Memory Access**

- **L1 Hit:**
  - \( H₁ = 0.985 \) or 98.5%
  - \( % = H₁ = 0.985 \) or 98.5%
  - Hit Access Time = 1 cycle
  - Stalls = 0

- **L1 Miss:**
  - \( (1-H₁) = 0.015 \) or 1.5%

- **L1 Miss, Clean**
  - Access Time = \( M + 1 = 51 \)
  - Stalls per access = \( M = 50 \)
  - Stall cycles = \( M \times (1-H₁) \times % \) clean
    - \( = 50 \times 0.0135 = 0.675 \) cycles

- **L1 Miss, Dirty**
  - Access Time = \( 2M + 1 = 101 \)
  - Stalls per access = \( 2M = 100 \)
  - Stall cycles = \( 2M \times (1-H₁) \times % \) dirty
    - \( = 100 \times 0.0015 = 0.15 \) cycles

**Stall Cycles Per Memory Access**

\[
\text{AMAT} = 1 + \text{Stall Cycles Per Memory Access} = 1 + 0.825 = 1.825 \text{ cycles}
\]

**Given Parameters:**
- \( H₁ = 98.5\% \)
- \( T₁ = 0 \) cycles
- \( M = 100 \) cycles
- Stalls on a hit in L₁
- L₁ Misses: 10% dirty 90% clean
- \( \text{CPI}_{\text{execution}} = 1.1 \)
- Memory accesses per instruction = 1.3

**Assuming:**
- Ideal access on a hit in L₁

\( M \) = Miss Penalty = 50 cycles
\( M + 1 = \) Miss Time = 50 + 1 = 51 cycles
\( \text{L1 access Time} = 1 \) cycle
\( H₁ = 0.985 \) or 98.5%
\( 1 - H₁ = 0.015 \) or 1.5%

\( \text{CPI} = \text{CPI}_{\text{execution}} + \text{Stall cycles per instruction} = 1.1 + 1.07 = 2.17 \)
Memory Access Tree Structure
For Separate Level 1 Caches, Write Back, With Write Allocate

CPU Memory Access

L1

% Instructions

% Instructions x (1 - Instruction H1 )

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

Instruction L1 Hit:
Hit Access Time = 1
Stalls = 0
% Instructions x Instruction H1

Data

% data x (1 - Data H1 )

Data L1 Miss:
Hit Access Time = 1
Stalls = 0
% data x Data H1

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

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

M = Miss Penalty = stall cycles per access resulting from missing in cache
M + 1 = Miss Time = Main memory access time
Data H1 = Level 1 Data Hit Rate
1 - Data H1 = Level 1 Data Miss Rate
Instruction H1 = Level 1 Instruction Hit Rate
1 - Instruction H1 = Level 1 Instruction Miss Rate
% instructions = Percentage or fraction of instruction fetches out of all memory accesses
% Data = Percentage or fraction of data accesses out of all memory accesses
% Clean = Percentage or fraction of data L1 misses that are clean
% Dirty = Percentage or fraction of data L1 misses that are dirty = 1 - % Clean
Improving Cache Performance: Multi-Level Cache

2 Levels of Cache: $L_1$, $L_2$

- **CPU**
  - Hit Rate = $H_1$
  - Hit Access Time = 1 cycle (No Stall)
  - Stalls for hit access = $T_1 = 0$

- **L1 Cache**
  - Ideal access on a hit in $L_1$

- **L2 Cache**
  - Local Hit Rate = $H_2$
  - Stalls per hit access = $T_2$
  - Hit Access Time = $T_2 + 1$ cycles

- **Main Memory**
  - Slower than $L_1$
  - But has more capacity

  Slower than $L_2$

  $L_1 = \text{Level 1 Cache}$
  $L_2 = \text{Level 2 Cache}$

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

**Goal of multi-level Caches:**
Reduce the effective miss penalty incurred by level 1 cache misses by using additional levels of cache that capture some of these misses. Thus hiding more main memory latency and reducing AMAT further.
Miss Rates For Multi-Level Caches

- **Local Miss Rate:** This rate is the number of misses in a cache level divided by the number of memory accesses to this level (i.e., those memory accesses that reach this level).
  
  \[ \text{Local Hit Rate} = 1 - \text{Local Miss Rate} \]

- **Global Miss Rate:** 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:
  
  \[ \text{Local Miss Rate} = \text{Global Miss Rate} = 1 - H_1 \]

- For level 2 since it only receives those accesses missed in 1:
  
  \[ \begin{align*}
  \text{Local Miss Rate} &= \text{Miss rate}_{L2} = 1 - H_2 \\
  \text{Global Miss Rate} &= \text{Miss rate}_{L1} \times \text{Miss rate}_{L2} \\
  &= (1 - H_1) \times (1 - H_2)
  \end{align*} \]

For Level 3, global miss rate?
2-Level Cache Performance
(Ignoring Write Policy)

CPU\text{time} = IC \times (CPI_{\text{execution}} + \text{Mem Stall cycles per instruction}) \times C

Mem Stall cycles per instruction = Mem accesses per instruction \times 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 =

\[ [\text{miss rate } L_1] \times [\text{Hit rate } L_2 \times \text{Hit time } L_2 + \text{Miss rate } L_2 \times \text{Memory access penalty}] =\]

\((1-H_1) \times H_2 \times T_2 + (1-H_1)(1-H_2) \times M\)

- L1 Miss, L2 Hit
- H1 = L1 Hit Rate
- T1 = stall cycles per L1 access hit
- H2 = Local L2 Hit Rate
- T2 = stall cycles per L2 access hit

- Here we assume T1 = 0 (no stall on L1 hit)

L1 Miss, L2 Miss: Must Access Main Memory

CPI = CPI_{\text{execution}} + (1 + \text{fraction of loads and stores}) \times \text{stall cycles per access}

= CPI_{\text{execution}} + (1 + \text{fraction of loads and stores}) \times (AMAT - 1)
2-Level Cache (Both Unified) Performance
Memory Access Tree  (Ignoring Write Policy)

CPU Stall Cycles Per Memory Access

CPU Memory Access

Assuming:
Ideal access on a hit in \(L_1\)
\(T_1 = 0\)

\(L_1\) Hit:
Hit Access Time = 1
Stalls = \(H_1 \times 0 = 0\)
(No Stall)

\(L_1\) Miss:
\(\% = (1 - H_1)\)

\(L_1\) Miss, \(L_2\) Hit:
Hit Access Time = \(T_2 + 1\)
Stalls per \(L_2\) Hit = \(T_2\)
Stalls = \((1 - H_1) \times H_2 \times T_2\)

\(L_1\) Miss, \(L_2\) Miss:
Access Time = \(M + 1\)
Stalls per access = \(M\)
Stalls = \((1 - H_1)(1 - H_2) \times \)\(M\)

Stall cycles per memory access = \((1 - H_1) \times H_2 \times T_2\) + \((1 - H_1)(1 - H_2) \times M\)

\(AMAT = 1 + (1 - H_1) \times H_2 \times T_2\) + \((1 - H_1)(1 - H_2) \times M\)

\(CPI = CPI_{\text{execution}} + (1 + \text{fraction of loads and stores}) \times \text{stall cycles per access}\)
\(= CPI_{\text{execution}} + (1 + \text{fraction of loads and stores}) \times (AMAT - 1)\)
Unified Two-Level Cache Example

- CPU with $CPI_{\text{execution}} = 1.1$ running at clock rate = 500 MHz
- 1.3 memory accesses per instruction.
- With two levels of cache (both unified)
- $L_1$ hit access time = 1 cycle (no stall on a hit, $T_1 = 0$), a miss rate of 5%
- $L_2$ hit access time = 3 cycles ($T_2 = 2$ stall cycles per hit) with local miss rate 40%,
- Memory access penalty, $M = 100$ cycles (stalls per access).

Find $CPI$ ...

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

With No Cache, \( CPI = 1.1 + 1.3 \times 100 = 131.1 \)

With single $L_1$, \( CPI = 1.1 + 1.3 \times .05 \times 100 = 7.6 \)

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

Stall cycles per memory access = \( (1-H_1) \times H_2 \times T_2 + (1-H_1)(1-H_2) \times M \)
\[
= 0.05 \times .6 \times 2 + 0.05 \times 0.4 \times 100 \\
= 0.06 + 2 = 2.06 \text{ cycles}
\]

AMAT = 2.06 + 1 = 3.06 cycles

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

\[
= 2.06 \times 1.3 = 2.678 \text{ cycles}
\]

$CPI = 1.1 + 2.678 = 3.778$

$\text{Speedup} = 7.6/3.778 = 2$

Compared to CPU with L1 only

$CPI = CPI_{\text{execution}} + (1 + \text{fraction of loads and stores}) \times \text{stall cycles per access}$

\[
= CPI_{\text{execution}} + (1 + \text{fraction of loads and stores}) \times (\text{AMAT} - 1)
\]
Memory Access Tree For 2-Level Cache (Both Unified) Example

CPU Stall Cycles Per Memory Access

**CPU Memory Access**

- **L1 Hit:**
  - Hit Access Time = 1
  - Stalls per L1 Hit = T1 = 0
  - Stalls = H1 x 0 = 0
  - (No Stall)

- **L1 Miss:**
  - (1-H1)= 0.05 or 5%

- **L1 Miss, L2 Hit:**
  - Hit Access Time = T2 + 1 = 3 cycles
  - Stalls per L2 Hit = T2 = 2 cycles
  - Stalls = (1-H1) x H2 x T2
  - = 0.03 x 2 = 0.06 cycles

- **L1 Miss, L2 Miss:**
  - Access Time = M + 1 = 100 + 1 = 101 cycles
  - Stalls per access = M = 100 cycles
  - Stalls = (1-H1)(1-H2) x M
  - = 0.02 x 100 = 2 cycles

**Stall cycles per memory access**

- = (1-H1) x H2 x T2 + (1-H1)(1-H2) x M
- = 0.06 x 2 = 2.06 cycles

**AMAT**

- = 1 + Stall cycles per memory access = 1 + 2.06 = 3.06 cycles

**Stall cycles per instruction**

- = (1 + fraction of loads/stores) x Stall Cycles per access
- = 1.3 x 2.06 = 2.678 cycles

**CPI**

- = CPI\textsubscript{execution} + Stall cycles per instruction
- = 1.1 + 2.678 = 3.778

**Given Parameters:**

- H1 = 95%  T1 = 0 cycles
- H2 = 60%  T2 = 2 cycles
- M = 100 cycles
- CPI\textsubscript{execution} = 1.1
- Memory accesses per instruction = 1.3

**Ideal access on a hit in L1**

- T1 = 0

**CPU Stall Cycles Per Memory Access**

- CPI = CPI\textsubscript{execution} + (1 + fraction of loads and stores) x stall cycles per access
- = CPI\textsubscript{execution} + (1 + fraction of loads and stores) x (AMAT – 1)
Memory Access Tree Structure For 2-Level Cache
(Separate Level 1 Caches, Unified Level 2) (Ignoring Write Policy)

CPU Memory Access

% Instructions
1 or 100%

% data

Instruction

Instruction L1 Hit:

Instruction L1 Miss:

Data

Data L1 Hit:

Data L1 Miss:

L1

L1

% Instructions = Percentage or fraction of instruction fetches out of all memory accesses
% Data = Percentage or fraction of data accesses out of all memory accesses

Exercise: In terms of the parameters below, complete the memory access tree and find the expression for stall cycles per memory access

For L1:
T1 = Stalls per hit access to level 1
Data H1 = Level 1 Data Hit Rate
Instruction H1 = Level 1 Instruction Hit Rate

For L2:
T2 = Stalls per access to level 2
H2 = Level 2 local hit Rate

M = Miss Penalty = stall cycles per access resulting from missing in cache level 2
M + 1 = Miss Time = Main memory access time

L2 Hit

L2 Miss

L2 Hit

L2 Miss

L2 Hit

L2 Miss
Write Policy For 2-Level Cache

- **Write Policy For Level 1 Cache:**
  - Usually Write through to Level 2.
  - Write allocate is used to reduce level 1 read misses.
  - Use write buffer to reduce write stalls to level 2.

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

- The above 2-level cache write policy results in **inclusive L2 cache** since the content of L1 is also in L2
  - Common in the majority of all CPUs with 2-levels of cache
  - As opposed to exclusive L1, L2 (e.g. AMD Athlon XP, A64)
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

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

L1 Miss:

L2: Write Back with Write Allocate

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

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

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

AMAT = 1 + Stall Cycles Per Memory Access
CPI = CPI_{execution} + (1 + fraction of loads and stores) x Stall Cycles per access

EECC551 - Shaaban
Two-Level (Both Unified) Cache Example With Write Policy

- CPU with $\text{CPI}_{\text{execution}} = 1.1$ running at clock rate = 500 MHz
- 1.3 memory accesses per instruction. Two levels of cache (both unified)

For $L_1$:
- Cache operates at 500 MHz (no stall on L1 Hit, $T_1 = 0$) with a miss rate of $1 - H_1 = 5\%$
- Write through to L2 with perfect write buffer with write allocate

For $L_2$:
- Hit access time = 3 cycles ($T_2 = 2$ stall cycles per hit) local miss rate $1 - H_2 = 40\%$
- Write back to main memory with write allocate
- Probability a cache block is dirty = $10\%$

- Memory access penalty, $M = 100$ cycles.
- Create memory access tree and find, stalls per memory access, AMAT, CPI.

Stall cycles per memory access = $(1 - H_1) \times H_2 \times T_2 + (1 - H_1) \times (1 - H_2) \times (\% \text{ clean} \times M + \% \text{ dirty} \times 2M)$

= $0.05 \times 0.6 \times 2 + 0.05 \times 0.4 \times (0.9 \times 100 + 0.1 \times 200)$

= $0.06 + 0.02 \times 110 = 0.06 + 2.2 = 2.26$

- $\text{AMAT} = 2.26 + 1 = 3.26$ cycles

Mem Stall cycles per instruction = Mem accesses per instruction $\times$ Stall cycles per access

$= 2.26 \times 1.3 = 2.938$ cycles

$\text{CPI} = 1.1 + 2.938 = 4.038 = 4$

$\text{CPI} = \text{CPI}_{\text{execution}} + (1 + \text{fraction of loads and stores}) \times (\text{AMAT} - 1)$
Memory Access Tree For Two-Level (Both Unified) Cache Example With Write Policy
L1: Write Through to L2, Write Allocate, With Perfect Write Buffer
L2: Write Back with Write Allocate

CPU Memory Access

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

L1 Miss:
(1-H1) x H2 = 0.05 x 0.6 = 0.03 or 3%

L2
L1 Miss, L2 Hit:
Hit Access Time = T2 + 1 = 3 cycles
Stalls per L2 Hit = T2 = 2 cycles
Stalls = (1-H1) x H2 x T2
= 0.03 x 2 = 0.06 cycles

L1 Miss, L2 Miss:
(1-H1) x (1-H2) x % dirty
= 0.02 x 0.1 = 0.002 or 0.2%

L1 Miss, L2 Miss, Clean
Access Time = M + 1 = 101 cycles
Stalls per access = M
Stall cycles = M x (1 -H1) x (1-H2) x % clean
= 100 x 0.018 = 1.8 cycles

L1 Miss, L2 Miss, Dirty
Access Time = 2M + 1 = 200 + 1 = 201 cycles
Stalls per access = 2M = 200 cycles
Stall cycles = 2M x (1-H1) x (1-H2) x % dirty
= 200 x 0.002 = 0.4 cycles

AMAT = 1 + Stall cycles per memory access
= 1 + 2.26 = 3.26 cycles
Stall cycles per instruction = (1 + fraction of loads/stores) x Stall Cycles per access
= 1.3 x 2.26 = 2.938 cycles
CPI = CPI_{execution} + Stall cycles per instruction
= 1.1 + 2.938 = 4.038
Memory Access Tree Structure For 2-Level Cache (Separate Level 1 Caches, Unified Level 2)
L1: Write Through to L2, Write Allocate, With Perfect Write Buffer  L2: Write Back with Write Allocate

CPU Memory Access

% Instructions  1 or 100%

Instruction

% data

Data

L1

Instruction L1 Hit:

Instruction L1 Miss:

Data L1 Hit:

Data L1 Miss:

L2

L2 Hit

L2 Miss

L2 Miss Clean

L2 Miss Dirty

% Instructions = Percentage or fraction of instruction fetches out of all memory accesses
% Data = Percentage or fraction of data accesses out of all memory accesses

For L1:
T1 = Stalls per hit access to level 1
Data H1 = Level 1 Data Hit Rate
Instruction H1 = Level 1 Instruction Hit Rate

For L2:
T2 = Stalls per access to level 2
H2 = Level 2 local hit rate

Exercise: In terms of the parameters below, complete the memory access tree and find the expression for stall cycles per memory access

L2 Miss

L2 Miss Clean

L2 Miss Dirty
3 Levels of Cache

CPU

L1 Cache

Hit Rate = H₁,
Hit Access Time = 1 cycle (No Stall)
Stalls for hit access = T₁ = 0

L2 Cache

Local Hit Rate = H₂
Stalls per hit access = T₂
Hit Access Time = T₂ + 1 cycles

L3 Cache

Local Hit Rate = H₃
Stalls per hit access = T₃
Hit Access Time = T₃ + 1 cycles

Main Memory

Slower than L₁
But has more capacity

Slower than L₂
But has more capacity

Slower than L₃

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

CPI = CPIₜₖᵢₜᵽₑₜₖᵲᵽᵽₑ + (1 + fraction of loads and stores) x stall cycles per access
= CPIₜₖᵢₜᵽₑₜₖᵲᵽₑ + (1 + fraction of loads and stores) x (AMAT – 1)
3-Level (All Unified) Cache Performance

CPUtime = IC \times (CPI_{\text{execution}} + \text{Mem Stall cycles per instruction}) \times C

\text{Mem Stall cycles per instruction} = \text{Mem accesses per instruction} \times \text{Stall cycles per access}

- For a system with 3 levels of cache, assuming no penalty when found in L_1 cache:

Stall cycles per memory access =

\[(\text{miss rate L}_1) \times [\text{Hit rate L}_2 \times \text{Hit time L}_2
\]
\[+ \text{Miss rate L}_2 \times (\text{Hit rate L}_3 \times \text{Hit time L}_3
\]
\[+ \text{Miss rate L}_3 \times \text{Memory access penalty}) ] =

(1-H_1) \times H_2 \times T_2

\hspace{1cm} + \ (1-H_1) \times (1-H_2) \times H_3 \times T_3

L_1 \text{ Miss, L}_2 \text{ Hit}

\hspace{1cm} + \ (1-H_1)(1-H_2)(1-H_3) \times M

L_2 \text{ Miss, L}_3 \text{ Hit}

\text{CPI} = CPI_{\text{execution}} + (1 + \text{fraction of loads and stores}) \times \text{stall cycles per access}
\hspace{1cm} = CPI_{\text{execution}} + (1 + \text{fraction of loads and stores}) \times (\text{AMAT} - 1)
**3-Level (All Unified) Cache Performance**

**Memory Access Tree (Ignoring Write Policy)**

**CPU Stall Cycles Per Memory Access**

---

**CPU Memory Access**

- **L1 Hit:**
  - Hit Access Time = 1
  - Stalls Per access = T1 = 0
  - Stalls = H1 x 0 = 0
  - (No Stall)

- **L1 Miss:**
  - % = (1 - H1)

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

- **L1 Miss, L2 Miss:**
  - % = (1 - H1)(1 - H2)

  - Stalls = (1 - H1)(1 - H2)(1 - H3) x M

- **L1 Miss, L2 Miss, L3 Hit:**
  - Hit Access Time = T3 + 1
  - Stalls per L2 Hit = T3
  - Stalls = (1 - H1) x (1 - H2) x H3 x T3

- **L1 Miss, L2 Miss, L3 Miss:**
  - Stalls = (1 - H1)(1 - H2)(1 - H3) x M

**Stall cycles per memory access**

\[
\text{Stall cycles per memory access} = (1 - H1) x H2 x T2 + (1 - H1) x (1 - H2) x H3 x T3 + (1 - H1)(1 - H2)(1 - H3) x M
\]

**AMAT**

\[
\text{AMAT} = 1 + \text{Stall cycles per memory access}
\]

**CPI**

\[
\text{CPI} = \text{CPI}_{\text{execution}} + (1 + \text{fraction of loads and stores}) \times \text{stall cycles per access}
\]

\[
= \text{CPI}_{\text{execution}} + (1 + \text{fraction of loads and stores}) \times (\text{AMAT} - 1)
\]
Three-Level (All Unified) Cache Example

- CPU with $\text{CPI}_{\text{execution}} = 1.1$ running at clock rate = 500 MHz
- 1.3 memory accesses per instruction.
- $L_1$ cache operates at 500 MHz (no stalls on a hit in $L_1$) with a miss rate of 5%.
- $L_2$ hit access time = 3 cycles ($T_2 = 2$ stall cycles per hit), local miss rate 40%.
- $L_3$ hit access time = 6 cycles ($T_3 = 5$ stall cycles per hit), local miss rate 50%.
- Memory access penalty, $M = 100$ cycles (stall cycles per access). Find CPI.

With No Cache,

$$\text{CPI} = 1.1 + 1.3 \times 100 = 131.1$$

With single $L_1$,

$$\text{CPI} = 1.1 + 1.3 \times 0.05 \times 100 = 7.6$$

With $L_1$, $L_2$

$$\text{CPI} = 1.1 + 1.3 \times (0.05 \times 0.6 \times 2 + 0.05 \times 0.4 \times 100) = 3.778$$

$$\text{CPI} = \text{CPI}_{\text{execution}} + \text{Mem Stall cycles per instruction}$$

Mem Stall cycles per instruction = Mem accesses per instruction $\times$ Stall cycles per access

Stall cycles per memory access

$$= (1-H_1) \times H_2 \times T_2 + (1-H_1) \times (1-H_2) \times H_3 \times T_3 + (1-H_1)(1-H_2) \times (1-H_3) \times M$$

$$= 0.05 \times 0.6 \times 2 + 0.05 \times 0.4 \times 0.5 \times 5 + 0.05 \times 0.4 \times 0.5 \times 100$$

$$= 0.06 + 0.05 + 1 = 1.11$$

$$\text{AMAT} = 1.11 + 1 = 2.11 \text{ cycles (vs. AMAT = 3.06 with L1, L2, vs. 5 with L1 only)}$$

$$\text{CPI} = 1.1 + 1.3 \times 1.11 = 2.54$$

Speedup compared to $L_1$ only $= 7.6/2.54 = 3$

Speedup compared to $L_1$, $L_2$ $= 3.778/2.54 = 1.49$

All cache levels are unified, ignoring write policy
Memory Access Tree For 3-Level Cache (All Unified) Example

CPU Memory Access

H1 = .95 or 95% 1 or 100%

L1 Hit:
Hit Access Time = 1
Stalls Per access = 0
Stalls = H1 x 0 = 0
(No Stall)
(1-H1) x H2
= 0.05 x .6
= 0.03 or 3%

L1 Miss:
(1-H1) = 0.05 or 5%

L1 Miss, L2 Hit:
Hit Access Time = T2 +1 = 3
Stalls per L2 Hit = T2 = 2
Stalls = (1-H1) x H2 x T2
= .05 x .6 x 2 = .06

L1 Miss, L2 Miss:
(1-H1)(1-H2) = .05 x .4 = .02 or 2%

L1 Miss, L2 Miss, L3 Hit:
Hit Access Time = T3 +1 = 6
Stalls per L2 Hit = T3 = 5
Stalls = (1-H1) x (1-H2) x H3 x T3
= .01 x 5 = .05 cycles

Stall cycles per memory access
= (1-H1) x H2 x T2 + (1-H1) x (1-H2) x H3 x T3 + (1-H1)(1-H2) (1-H3)x M
= .06 + .05 + 1 = 1.11

AMAT = 1 + Stall cycles per memory access
= 1 + 1.11 = 2.11 cycles

Stall cycles per instruction = (1 + fraction of loads/stores) x Stall Cycles per access
= 1.3 x 1.11 = 1.443 cycles

CPI = CPI_{execution} + Stall cycles per instruction
= 1.1 + 1.443 = 2.543

L1 Miss, L2 Miss, L3 Miss:
Miss Penalty = M = 100
Stalls = (1-H1)(1-H2)(1-H3) x M
= .01 x 100 = 1 cycle

Exercise: Create the memory access tree for 3-level cache where level 1 is split and Levels 2, 3 are unified once ignoring write policy and once with write back for L3. Find the expression for memory stalls per access for either case.