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.

4th Edition: Chapter 5.1, Appendix C.1-C.3 (3rd Edition Chapter 5.1-5.4)
Memory Hierarchy: Motivation

Processor-Memory (DRAM) Performance Gap

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

Memory Access Latency: The time between a memory access request is issued by the processor and the time the requested information (instructions or data) is available to the processor.

![](chart.png)

- Ideal Memory Access Time (latency) = 1 CPU Cycle
- Real Memory Access Time (latency) >> 1 CPU cycle

(Review from 550)
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:

  \[
  \text{Minimum CPU memory stall cycles} = \frac{\text{CPU cycles} + \text{Memory Accesses}}{\text{CPU speed (MHZ)}} - 1
  \]

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

(Review from 550)

Ideal Memory Access Time (latency) = 1 CPU Cycle
Real Memory Access Time (latency) >> 1 CPU cycle

Or more 200+ Cycles

EECC551 - Shaaban
Memory Access Latency Reduction & Hiding Techniques

Memory Latency Reduction Techniques:  
- Faster Dynamic RAM (DRAM) Cells: Depends on VLSI processing technology.
- Wider Memory Bus Width: Fewer memory bus accesses needed (e.g., 128 vs. 64 bits)
- Multiple Memory Banks:  
  - At DRAM chip level (SDR, DDR, SDRAM), module or channel levels.
- Integration of Memory Controller with Processor: e.g., AMD’s processor architecture + Intel’s i7
- New Emerging Faster RAM Technologies: e.g., Magnetoresistive Random Access Memory (MRAM)

Memory Latency Hiding Techniques:  
- Memory Hierarchy: One or more levels of smaller and faster memory (SRAM-based cache) on- or off-chip that exploit program access locality to hide long main memory latency.
- Pre-Fetching: Request instructions and/or data from memory before actually needed to hide long memory access latency.

Get it from main memory into cache before you need it!

Addressing The CPU/Memory Performance Gap:  
Reduce it!  
Hide it!  
What about dynamic scheduling?
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. For Ideal Memory: Memory Access Time or latency = 1 CPU cycle

• To hide long memory access latency, the memory hierarchy is organized into several levels of memory with the smaller, faster SRAM-based memory levels closer to the CPU: registers, then primary Cache Level (L₁), then additional secondary cache levels (L₂, L₃…), then DRAM-based 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.
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

Interface:
SCSI, RAID, IDE, 1394
80G-300G

Cache Level(s)

Main Memory

Magnetic Disc

Optical Disk or Magnetic Tape

CPU

Faster Access
Time

Closer to CPU Core

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

(Virtual Memory)

(Review from 550)
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 → Program Working Set

• Two Types of access locality:

  1 – 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

  2 – 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)

Cache utilizes faster memory (SRAM)
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. (i.e Cache)

Locality in program memory access  →  Program Working Set
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): $L_1$ 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
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:
    - (S)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 + Time to deliver the missed block to the processor
- **Hit Time << Miss Penalty**

(Review from 550)

Typical Cache Block (or line) Size: 16-64 bytes

Cache Hit if block is found in cache

Lower Level Memory
  e.g main memory

Level 1 (L1) Cache
  Blk X

From Processor
  (Store)

To Processor
  (Fetch/Load)

Upper Level Memory
  e.g cache

Hit rate for level one cache = $H_1$

Miss rate for level one cache = $1 - Hit rate = 1 - H_1$

Ideally = 1 Cycle

Miss

Or Miss Time

M Stall cycles on a miss

EECC551 - Shaaban

#10  lec # 8  Spring 2011  4-13-2011
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.
Basic Cache Design & Operation Issues

- **Q1:** Where can a block be placed in cache? *(Block placement strategy & Cache organization)*
  - Fully Associative, Set Associative, Direct Mapped.  
    - Very complex  
    - Most common  
    - Simple but suffers from conflict misses

- **Q2:** How is a block found if it is in cache? *(Block identification)*
  - Tag/Block. *(Tag Matching)*

- **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.

(Review from 550)

4th Edition: Appendix C.1 (3rd Edition Chapter 5.2)

Not covered in 550 will be covered here

EECC551 - Shaaban
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:
   \[
   \text{index} = \text{(Block address)} \mod \text{(Number of blocks in cache)}
   \]

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

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:
   \[
   \text{index} = \text{(Block address)} \mod \text{(Number of sets in cache)}
   \]
   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

(i.e low three bits of block address)

8 cache block frames

Here four blocks in memory map to the same cache block frame

32 memory blocks cacheable

Limitation of Direct Mapped Cache: Conflicts between memory blocks that map to the same cache block frame may result in conflict cache misses

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

Index size = Log₂ 8 = 3 bits
4KB Direct Mapped Cache Example

1K = $2^{10} = 1024$ Blocks
Each block = one word (4 bytes)

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 bits of block address

Tag Matching

Hit or Miss Logic (Hit or Miss?)

Direct mapped cache is the least complex cache organization in terms of tag matching and Hit/Miss Logic complexity

Hit Access Time = SRAM Delay + Hit/Miss Logic Delay

(Review from 550)
64KB Direct Mapped Cache Example

Tag field (16 bits)

`4K = 2^{12} = 4096` blocks
Each block = four words = 16 bytes

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

Typical cache Block or line size: 32-64 bytes

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

Hit or miss?

Block Address = 28 bits

Tag = 16 bits

Index = 12 bits

Block offset = 4 bits

Mapping

Hit Access Time = SRAM Delay + Hit/Miss Logic Delay

EECC551 - Shaaban
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

A cache with a total of 8 cache block frames shown above
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) = index = 100

**2-way Set associative:**
- block 12 can go anywhere in set 0
  - (12 mod 4) = index = 00

---

8 Block Frames

Cache

Block no.
0 1 2 3 4 5 6 7

32 Block Frames

Memory

Block no.
0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1

This example cache has eight block frames and memory has 32 blocks.
4K Four-Way Set Associative Cache: MIPS Implementation Example

Address

Tag Field (22 bits)

Block Offset Field (2 bits)

Index Field (8 bits)

Set Number

SRAM

Hit/ Miss Logic

Mapping

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

Mapping Function:

Cache Set Number = index = (Block address) MOD (256)

1024 block frames
Each block = one word
4-way set associative
1024 / 4 = $2^8 = 256$ 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

Hit Access Time = SRAM Delay + Hit/Miss Logic Delay

(Review from 550)

EECC551 - Shaaban

Typically, primary or Level 1 (L1) cache is 2-8 way set associative
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)

No index/mapping function for fully associative cache

EECC551 - Shaaban

(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)

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

2 – **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.

3 – **First In, First Out (FIFO):**
- Because LRU can be complicated to implement, this approximates LRU by determining the oldest block rather than LRU
## 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></td>
<td>LRU</td>
<td>Random</td>
<td>LRU</td>
</tr>
<tr>
<td>Size (Nominal)</td>
<td></td>
<td></td>
<td></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 \((L_1)\) cache is used for both instructions and data.

- **Separate instruction/data Level 1 caches (Harvard Memory Architecture):**
  The level 1 \((L_1)\) cache is split into two caches, one for instructions (instruction cache, \(L_1\) I-cache) and the other for data (data cache, \(L_1\) D-cache).

---

**Diagram:**

- **Unified Level 1 Cache (Princeton Memory Architecture):**
  - Processor
  - Control
  - Datapath
  - Registers
  - Unified Level One Cache \(L_1\)

- **Separate (Split) Level 1 Caches (Harvard Memory Architecture):**
  - Processor
  - Control
  - Datapath
  - Registers
  - \(L_1\) I-cache
  - \(L_1\) D-cache

---

(Review from 550)

Split Level 1 Cache is more preferred in pipelined CPUs to avoid instruction fetch/Data access structural hazards.
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.
- **Average 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 = (AMAT -1)**
- **For ideal memory:** AMAT = 1 cycle, this results in zero memory stall cycles.
- **Memory stall cycles per average instruction =**
  
  \[ \text{Number of memory accesses per instruction} \downarrow \times \text{Memory stall cycles per average memory access} \]

  \[= \left( 1 + \text{fraction of loads/stores} \right) \times (\text{AMAT} - 1) \]

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

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

cycles = CPU cycles
Cache Performance:
Single Level L1 Princeton (Unified) Memory Architecture

CPU time = Instruction count \times CPI \times Clock cycle time

CPI_{\text{execution}} = CPI with ideal memory

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

Mem Stall cycles per instruction =

Memory accesses per instruction \times Memory stall cycles per access

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

Cache Hit Rate = H_1 \quad \text{Miss Rate} = 1 - H_1

Memory stall cycles per memory access = Miss rate \times Miss penalty = (1 - H_1) \times M

AMAT = 1 + Miss rate \times Miss penalty = 1 + (1 - H_1) \times M

Memory accesses per instruction = (1 + \text{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_{\text{execution}} + (1 + \text{fraction of loads/stores}) \times (1 - H_1) \times M

AMAT = 1 + (1 - H_1) \times M
Memory Access Tree:
For Unified Level 1 Cache

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 = \( H1 \times 1 + (1 - H1) \times (M + 1) = 1 + M \times (1 - H1) \)

Stall Cycles Per Access = AMAT - 1 = \( M \times (1 - H1) \)

CPI = CPI_{execution} + (1 + 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.
- CPI_{execution} = 1.1 (i.e. base CPI with ideal memory)
- 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.

CPI = CPI_{execution} + mem stalls per instruction

Mem Stalls per instruction = Mem accesses per instruction \times Miss rate \times Miss penalty

\text{Mem accesses per instruction} = 1 + .3 = 1.3

Mem Stalls per memory access = (1- H1) \times M = .015 \times 50 = .75 \text{ cycles}

AMAT = 1 + .75 = 1.75 \text{ cycles}

Mem Stalls per instruction = 1.3 \times .015 \times 50 = 0.975

CPI = 1.1 + .975 = 2.075

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

M = Miss Penalty = 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 = \( \frac{(CPI_{old} \times C_{old})}{(CPI_{new} \times C_{new})} \)
            = \( \frac{2.075 \times 2}{3.05} \) = 1.36

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

→ CPUs with higher clock rate, have more cycles per cache miss and more memory impact on CPI.
Cache Performance:

Single Level L1 Harvard (Split) 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:

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

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

\[
M = \text{Miss Penalty} = \text{stall cycles per access to main memory resulting from missing in cache}
\]

\[
\text{CPI}_\text{execution} = \text{base CPI with ideal memory}
\]

\[
\text{Miss rate} = 1 - \text{data H1}
\]

\[
\text{Miss rate} = 1 - \text{instruction H1}
\]

As assumption
In reality, even a hit may result in few stalls (2-3)

This is one method to find stalls per instruction
another method is shown in next slide
Memory Access Tree
For Separate Level 1 Caches

CPU Memory Access

- % Instructions
- % Data

Instruction

- % Instructions
- % Instructions x (1 - 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
- % 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

Assuming:
Ideal access on a hit, no stalls

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
Split L1 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.
  - \( \text{CPI}_{\text{execution}} = 1.1 \) (i.e base CPI with ideal memory)
  - 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 stalls per access, AMAT and CPI using this cache?

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

Memory Stall cycles per instruction = \( \text{Instruction Fetch Miss rate x Miss Penalty} + \text{Data Memory Accesses Per Instruction x Data Miss Rate x Miss Penalty} \)

Memory Stall cycles per instruction = \( \frac{0.5}{100} \times 200 + \frac{0.3}{6/100} \times 200 = 1 + 3.6 = 4.6 \) cycles
Stall cycles per average memory access = \( \frac{4.6}{1.3} = 3.54 \) cycles
AMAT = \( 1 + 3.54 = 4.54 \) cycles
\[
\text{CPI} = \text{CPI}_{\text{execution}} + \text{mem stalls per instruction} = 1.1 + 4.6 = 5.7 \text{ cycles}
\]

- What is the miss rate of a single level unified cache that has the same performance?

4.6 = \( 1.3 \times \text{Miss rate x 200} \) which gives a miss rate of 1.8% for an equivalent unified cache

- How much faster is the CPU with ideal memory?

The CPU with ideal cache (no misses) is \( \frac{5.7}{1.1} = 5.18 \) times faster
With no cache at all the CPI would have been \( 1.1 + 1.3 \times 200 = 261.1 \) cycles !!
Memory Access Tree For Separate Level 1 Caches Example

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

Instruction

% Instructions = 0.769 or 76.9%

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

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 = 76.92 cycles

Stall Cycles Per Access = % Instruction 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

CPI = CPIexecution + Stall cycles per instruction = 1.1 + 4.6 = 5.7

Data

% Data = 0.231 or 23.1%

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

Data L1 Miss:
Access Time = M + 1 = 201
Stalls Per access = M = 200
Stalls = %data x (1 - Data H1 ) x M
= 0.01385 x 0.06 x 200 = 2.769 cycles

(Review from 550)

For Last Example

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 = 0.995 or 99.5%  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
   - Can be reduced by increasing cache block size and pre-fetching

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).
   - Can be reduced by increasing total cache size

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.
   - Can be reduced by increasing cache associativity
The 3 Cs of Cache:
Absolute Steady State Miss Rates (SPEC92)

(For Unified L1 Cache)

- Compulsory
  
  i.e For fully associative (no conflict misses)
The 3 Cs of Cache:

Relative Steady State Miss Rates (SPEC92)
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.
Cache Write Strategies

1 **Write Through**: Data is written to both the cache block and to a block of main memory. (i.e written though to 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 Block Frame for Write-Back Cache:

- **D** = Dirty or Modified Status Bit
  - 0 = clean
  - 1 = dirty or modified

- **V** = Valid Bit

Data Tag

The updated cache block is marked as modified or dirty.
Cache Write Strategies:

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

Write Through

Write Hit Operation

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 to cache and also to memory

Write Back

Write Hit Operation

Write to cache

Set modified/dirty bit to 1 to indicate that cache block has been modified (i.e., block is dirty)
No write to memory

Write back to memory when replaced in cache

Cache Write Hit = block to be modified is found in cache

Cache Write Miss = block to be modified is not in cache

#40 lect #8 Spring 2011 4-13-2011
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:** *(Bring old block to cache then update it)*
  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:**
  The block is modified in the lower level (lower cache level, or main memory) and not loaded (written or updated) into cache.

  i.e A cache block frame is **not allocated** for the block to be modified (written-to)

  *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.

  Cache Write Miss = Block to be modified is not in cache

  Allocate = Allocate or assign a cache block frame for written data
Write Back Cache With Write Allocate: Cache Miss Operation
(read or write miss)

Block to be replaced is clean

Miss Penalty = M

1. CPU reads or writes to block in cache

2. Block to be replaced is clean
   - i.e. D was = 0
   - Replaced (old) block is discarded since it’s clean

3. Read missed block from memory
   - Penalty = M

Thus:
Total Miss Penalty = M + M = 2M

Block to be replaced is dirty (modified)

1. Write back modified block being replaced to memory
   - i.e. D was = 1
   - Penalty = M

2. Read missed block from memory
   - Penalty = M

3. CPU reads or writes to block in cache

4. Write replaced modified block to memory
   - Penalty = M

Thus:
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

- **Instruction Fetch + Loads**
  - **Unified $L_1$**

- **CPU Memory Access**
  - **Read**
    - % reads x 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
      - **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

**Assuming:**
Ideal access on a read hit, no stalls

**Exercise:**
Create memory access tree for split level 1

**Stall Cycles Per Memory Access =**

\[
AMAT = 1 + \text{% reads x (1 - H1) x M} + \text{% write x M}
\]

**CPI =**

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

**Stall Cycles per access = AMAT - 1**

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

- M = Miss Penalty
- M + 1 = Miss Time = Main memory access time
- H1 = Level 1 Hit Rate
- 1 - H1 = Level 1 Miss Rate
Reducing Write Stalls For Write Though Cache Using 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):
    \[
    \text{Stall Cycles Per Memory Access} = \% \text{ reads } \times (1 - H1) \times M
    \]
    (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):
    \[
    \text{Stall Cycles/Memory Access} = (\% \text{ reads } (1 - H1) + \% \text{ write stalls not eliminated }) \times M
    \]
**Write Through Cache Performance Example**

- A CPU with $\text{CPI}_{\text{execution}} = 1.1$ 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.

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

<table>
<thead>
<tr>
<th>Reads</th>
<th>Writes</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\frac{1.15}{1.3} = 88.5%$</td>
<td>$\frac{.15}{1.3} = 11.5%$</td>
</tr>
</tbody>
</table>

1. **With No Write Buffer**: Stall on all writes
   - Mem Stalls/ instruction = $1.3 \times 50 \times (88.5\% \times 1.5\% + 11.5\%) = 8.33$ cycles
   - CPI = $1.1 + 8.33 = 9.43$

2. **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 = $1.1 + 0.86 = 1.96$

3. **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 $L_1$
Write Back, With Write Allocate

CPU Memory Access

- **L1 Hit:**
  - $\% = H_1$
  - Hit Access Time = 1
  - Stalls = 0

- **L1 Miss**
  - $1 - H_1$
  - Stalls per access = $M$
  - Stall cycles = $M \times (1 - H_1) \times \%$ clean

**L1 Miss, Clean**
- Access Time = $M + 1$
- Stalls per access = $M$
- Stall cycles = $M \times (1 - H_1) \times \%$ clean

**L1 Miss, Dirty**
- Access Time = $2M + 1$
- Stalls per access = $2M$
- Stall cycles = $2M \times (1 - H_1) \times \%$ dirty

- **2M needed to:**
  - Write (back) Dirty Block
  - Read new block
  - (2 main memory accesses needed)

**Stall Cycles Per Memory Access**

\[
(1 - H_1) \times (M \times \% \text{ clean} + 2M \times \% \text{ dirty})
\]

**AMAT**

\[
1 + \text{Stall Cycles Per Memory Access}
\]

**CPI**

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

- **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

assuming:

- Ideal access on a hit, no stalls
- One access to main memory to get needed block

*Unified L1**

EECC551 - Shaaban

#46  lec # 8  Spring 2011  4-13-2011
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 = $(1-H_1) \times (M \times \% \text{ clean} + 2M \times \% \text{ dirty})$

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%
- Hit Access Time = 1
- 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 x (1 - H₁) x % clean
  = 50 x 0.0135 = 0.675 cycles

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

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

Stall Cycles Per Memory Access = 0.675 + 0.15 = 0.825 cycles

AMAT = 1 + Stall Cycles Per Memory Access = 1 + 0.825 = 1.825 cycles

Stall cycles per instruction = (1 + fraction of loads/stores) x Stall Cycles per access = 1.3 x 0.825 = 1.07 cycles

CPI = CPI\text{\_execution} + Stall cycles per instruction = 1.1 + 1.07 = 2.17

Given Parameters:
- H₁ = 98.5%  
  T₁ = 0 cycles
- M = 50 cycles  
  Stalls on a hit in L₁
- L₁ Misses: 10% dirty 90% clean
- CPI\text{\_execution} = 1.1
- Memory accesses per instruction = 1.3

**For Last Example**

Unified

**L₁**

Assuming:
- Ideal access on a hit in L₁

2M needed to Write Dirty Block and Read new block

M = Miss Penalty = 50 cycles
M + 1 = Miss Time = 50 + 1 = 51 cycles  
L₁ access Time = 1 cycle
H₁ = 0.985 or 98.5%
1 - H₁ = 0.015 or 1.5%
Memory Access Tree Structure
For Separate Level 1 Caches, Write Back, With Write Allocate
(AKA Split)

CPU Memory Access

Instruction

- % Instructions
- % Instructions x 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 (1 - Instruction H1)

Data

- % Data
- % Data x Data H1
- Data L1 Miss:
  - Access Time = M + 1
  - Stalls per access = M
  - Stall cycles = M x % data x (1 - Data H1) x % clean
- Data L1 Hit:
  - Hit Access Time = 1
  - Stalls = 0
  - % Data x (1 - 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

Exercise: Find expression for: Stall cycles per average memory access, AMAT

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
Instruction H1 = Level 1 Instruction Hit 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

Assuming:
Ideal access on a hit in L1

EECC551 - Shaaban

#49 lec #8 Spring 2011 4-13-2011
Improving Cache Performance: Multi-Level Cache

2 Levels of Cache: \( L_1, L_2 \)

**Basic Design Rule for \( L_1 \) Cache:**
- **K.I.S.S**
  - (e.g. low degree of associativity and capacity to keep it fast)
  - 3-4 cycles access time
  - 2-8 way set associative

**L_2** has slower access time than \( L_1 \), but has more capacity and higher associativity
- 10-15 cycles access time
- 8-16 way set associative

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

**L_1 Cache**
- Ideal access on a hit in \( L_1 \)

**L_2 Cache**
- Local Hit Rate = \( H_2 \)
- Stalls per hit access = \( T_2 \)
- Hit Access Time = \( T_2 + 1 \) cycles

**Main Memory**
- Slower (longer access time) than \( L_2 \)

**Memory access penalty, \( M \)**
- (stalls per main memory access)
- Access Time = \( M +1 \)
- Typically 200+ Cycles

**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:

  \[
  \text{Local Miss Rate} = \text{Miss rate}_{L2} = 1 - H_2
  \]

  \[
  \text{Global Miss Rate} = \text{Miss rate}_{L1} \times \text{Local Miss rate}_{L2}
  = (1 - H_1) \times (1 - H_2)
  \]

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

\[
\text{CPUtime} = IC \times (\text{CPI}_{\text{execution}} + \text{Mem Stall cycles per instruction}) \times C
\]

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 L1 cache: \((T_1 = 0)\)

Average stall cycles per memory access =

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

\[
\begin{align*}
\text{L1 Miss, L2 Hit} & : H_1 = L1 \text{ Hit Rate} \\
\text{T1} & = \text{stall cycles per L1 access hit} \\
\text{H2} & = \text{Local L2 Hit Rate} \\
\text{T2} & = \text{stall cycles per L2 access hit} \\
\end{align*}
\]

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

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

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

Memory Access Tree (Ignoring Write Policy)

CPU Stall Cycles Per Memory Access

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

L1 Miss:
Hit Access Time = 1
Stalls = H1 \times 0 = 0
(No Stall)

L1 Hit:
Hit Access Time = 1
Stalls = H1 \times 0 = 0
(No Stall)

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

L1 Miss, L2 Miss:
Access Time = M + 1
Stalls per access = M
Stalls = (1-H1)(1-H2) \times M

Global Miss Rate for Level 2

Assuming:
Ideal access on a hit in L1
T1 = 0

Global Hit Rate for Level 2
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 ...

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

With No Cache, \[ \text{CPI} = 1.1 + 1.3 \times 100 = 131.1 \]

With single L_1, \[ \text{CPI} = 1.1 + 1.3 \times 0.05 \times 100 = 7.6 \]

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

Stall cycles per memory access = (1-H_1) x H_2 x T_2 + (1-H_1)(1-H_2) x M
\[ = 0.05 \times 0.6 \times 2 + 0.05 \times 0.4 \times 100 \]
\[ = 0.06 + 2 = 2.06 \text{ cycles} \]

AMAT = 2.06 + 1 = 3.06 cycles

Mem Stall cycles per instruction = Mem accesses per instruction x Stall cycles per access
\[ = 2.06 \times 1.3 = 2.678 \text{ cycles} \]

\[ \text{CPI} = 1.1 + 2.678 = 3.778 \]

Speedup = 7.6/3.778 = 2 Compared to CPU with L_1 only
Memory Access Tree For 2-Level Cache (Both Unified) Example

CPU Stall Cycles Per Memory Access

CPU Memory Access

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

L1 Hit:
Hit Access Time = 1
Stalls per L1 Hit = T1 = 0
(No Stall)

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 + 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_{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_{execution} = 1.1
Memory accesses per instruction = 1.3

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

CPU Memory Access

% 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:
- $T_1 =$ Stalls per hit access to level 1
- $H_1 =$ Level 1 Data Hit Rate
- $1 - H_1 =$ Level 1 Data Miss Rate
- $H_1 =$ Level 1 Instruction Hit Rate
- $1 - H_1 =$ Level 1 Instruction Miss Rate

For L2:
- $T_2 =$ Stalls per access to level 2
- $H_2 =$ Level 2 local hit Rate
- $1 - H_2 =$ Level 2 local miss rate

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

Exercise: In terms of the parameters below, complete the memory access tree and find the expression for stall cycles per memory access.
Common Write Policy For 2-Level Cache

- **Write Policy For Level 1 Cache:**
  - *Usually Write through to Level 2.* (not write through to main memory just to L2)
  - 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)

As if we have a single level of cache with one portion (L1) is faster than remainder (L2)

i.e what is in L1 is not duplicated in L2
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

Assuming:
Ideal access on a hit in L1
T1 = 0

Unified
L1

Unified
L2

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

(H1)

L1 Miss:

(1-H1)

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

(1-H1) x H2

L1 Miss, L2 Hit:

(1-H1) x (1-H2)

L1 Miss, L2 Miss

Stalls = (1-H1) x (1-H2) x % dirty

Global Miss Rate for L2

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)

AMAT = 1 + Stall Cycles Per Memory Access
CPI = CPI_{execution} + (1 + fraction of loads and stores) x Stall Cycles per access
Two-Level (Both Unified) Cache Example With Write Policy

- CPU with CPI\textsubscript{execution} = 1.1 running at clock rate = 500 MHz
- 1.3 memory accesses per instruction. Two levels of cache (both unified)
- For L\textsubscript{1}:
  - Cache operates at 500 MHz (no stall on L1 Hit, T1 =0) with a miss rate of 1-H1 = 5%
  - Write though to L\textsubscript{2} with perfect write buffer with write allocate
- For L\textsubscript{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.
- Create memory access tree and find, stalls per memory access, AMAT, CPI.
- Stall cycles per memory access =
  \[
  (1-H1) \times H2 \times T2 + \frac{(1-H1) \times (1-H2) \times (\text{% clean} \times M + \text{% dirty} \times 2M)}{}
  \]
  \[
  = .05 \times .6 \times 2 + .05 \times .4 \times (.9 \times 100 + .1 \times 200)
  \]
  \[
  = .06 + 0.02 \times 110 = .06 + 2.2 = 2.26
  \]
- AMAT = 2.26 + 1 = 3.26 cycles
- Mem Stall cycles per instruction = Mem accesses per instruction \times Stall cycles per access
  \[
  = 2.26 \times 1.3 = 2.938 \text{ cycles}
  \]
  \[
  CPI = 1.1 + 2.938 = 4.038 = 4
  \]
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 Hit:
Hit Access Time = 1
Stalls Per access = 0

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

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 % clean = 0.02 x 0.9 = 0.018 or 1.8%

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

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 = 0.06 + 1.8 + 0.4 = 2.26 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

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

Given Parameters:
- H1 = 95%  T1 = 0 cycles
- H2 = 60%  T2 = 2 cycles
- M = 100 cycles  Stalls on a hit
- L2 Misses: 10% dirty 90% clean
- CPI_{execution} = 1.1
- Memory accesses per instruction = 1.3

EECC551 - Shaaban  #60 lec #8 Spring 2011 4-13-2011
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 = 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:
- \( T_1 = \) Stalls per hit access to level 1
- \( H_1 = \) Level 1 Data Hit Rate
- Instruction \( H_1 = \) Level 1 Instruction Hit Rate

For L2:
- \( T_2 = \) Stalls per access to level 2
- \( H_2 = \) 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
3 Levels of Cache

CPU

L1 Cache

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

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

L2 Cache

Slower than \( L_1 \),
But has more capacity
and higher associativity

10-15 cycles access time
8-16 way set associative

Local Hit Rate = \( H_3 \)
Stalls per hit access = \( T_3 \)
Hit Access Time = \( T_3 + 1 \) cycles

L3 Cache

Slower than \( L_2 \),
But has more capacity
and higher associativity

30-60 cycles access time
16-64 way set associative

Main Memory

Slower than \( L_3 \)

\( L_1 = \) Level 1 Cache
\( L_2 = \) Level 2 Cache
\( L_3 = \) Level 3 Cache

Memory access penalty, \( M \)
(stalls per main memory access)
Access Time = \( M + 1 \)
Typically 200+ Cycles

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)
CPU 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 3 levels of cache, assuming no penalty when found in L1 cache: \( T_1 = 0 \)

Stall cycles per memory access =

\[
\begin{align*}
\text{L1 Hit (here we assume } T_1 = 0) \\
(1-H1) \times H2 \times T2 \\
+ (1-H1) \times (1-H2) \times H3 \times T3 \\
+ (1-H1)(1-H2)(1-H3) \times M
\end{align*}
\]

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)
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
Hit: Stalls Per access = T1 = 0
Stalls= \( H1 \times 0 = 0 \)
\( \text{No Stall} \)

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)

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

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 = 1 + Stall cycles per memory access

CPI = CPI_{execution} + (1 + fraction of loads and stores) x stall cycles per access
= CPI_{execution} + (1 + fraction of loads and stores) x (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 $= \begin{align*} & (1-H_1) x H_2 x T_2 + (1-H_1) x (1-H_2) x H_3 x T_3 + (1-H_1)(1-H_2)(1-H_3)x M \\ &= 0.05 \times 0.6 \times 2 + 0.05 \times 0.4 \times 0.5 \times 5 + 0.05 \times 0.4 \times 0.5 \times 100 \\ &= 0.06 + 0.05 + 1 = 1.11 \end{align*}$

AMAT = 1.11 + 1 = 2.11 cycles (vs. AMAT = 3.06 with $L_1$, $L_2$, vs. 5 with $L_1$ 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%**

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

**L1 Miss:**
- Hit Access Time = T3 + 1 = 6
- (1-H1) x H2
- = 0.05 x .6
- = 0.03 or 3%

**L1 Miss, L2 Miss:**
- Hit Access Time = T2 + 1 = 3
- Stalls per L2 Hit = T2 = 2
- Stalls = (1-H1) x H2 x T2
- = 0.05 x .6 x 2 = 0.06

**L1 Miss, L2 Hit:**
- Hit Access Time = T2 + 1 = 3
- Stalls per L2 Hit = T2 = 2
- Stalls = (1-H1) x H2 x T2
- = 0.05 x .6 x 2 = 0.06

**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
- = 0.05 x 5 = 0.25 cycles

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

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

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

**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.