**Cache Measures**

- **Hit rate**: fraction found in that level
  - So high that usually we focus on **Miss rate**
- **Average memory-access time**
  - \( AMAT = \text{Hit time} + \text{Miss rate} \times \text{Miss penalty} \)
- **Miss penalty**: time to replace a block from lower level, including time to forward to CPU
  - **access time**: time to lower level
    - = \( f(\text{latency to lower level}) \)
  - **transfer time**: time to transfer block
    - = \( f(\text{BW between upper & lower levels}) \)
Cache performance - Impact on CPU

♦ Miss-oriented Approach to Memory Access:

\[ \text{CPUtime} = IC \times \left( \frac{\text{CPI}}{\text{Execution}} + \frac{\text{MemAccess}}{\text{Inst}} \times \text{MissRate} \times \text{MissPenalty} \right) \times \text{CycleTime} \]

- CPI\text{Execution} includes ALU and Memory instructions

♦ Separating out Memory component entirely

- AMAT = Average Memory Access Time
- CPI\text{ALUOps} does not include memory instructions

\[ \text{CPUtime} = IC \times \left( \frac{\text{AluOps}}{\text{Inst}} \times \frac{\text{CPI}_{\text{AluOps}}}{\text{Inst}} \times \frac{\text{MemAccess}}{\text{Inst}} \times \text{AMAT} \right) \times \text{CycleTime} \]

\[ \text{AMAT} = \text{HitTime} + \text{MissRate} \times \text{MissPenalty} \]

\[ = \left( \frac{\text{HitTime}_{\text{Inst}}}{\text{Inst}} + \frac{\text{MissRate}_{\text{Inst}}}{\text{Inst}} \times \frac{\text{MissPenalty}_{\text{Inst}}}{\text{Inst}} \right) \times \text{Freq}_{\text{Inst}} + \text{F}_{\text{data}} + \left( \frac{\text{HitTime}_{\text{data}}}{\text{Inst}} + \frac{\text{MissRate}_{\text{data}}}{\text{Inst}} \times \frac{\text{MissPenalty}_{\text{data}}}{\text{Inst}} \right) \times \text{Freq}_{\text{data}} \]

Caches in the Pipeline

ECE568/Koren Part.13
Adapted from UCB and other sources
Copyright UCB & Morgan Kaufmann
Unified vs Split Caches

- Unified vs Separate I&D

- Example:
  - 16KB I&D: Inst miss_rate=0.64%, Data miss_rate=6.47%
  - 32KB unified: Aggregate miss rate=1.99%

- Which is better (ignore L2 cache misses)?
  - Assume 33% data ops => 75% accesses for instructions (1.0/1.33)
  - hit_time=1, miss_time=50
  - Note that data hit has 1 stall for unified cache (only one port)

\[ \text{AMAT}_{\text{Harvard}} = 75\% \times (1+0.64\% \times 50)+25\% \times (1+6.47\% \times 50) = 2.05 \]
\[ \text{AMAT}_{\text{Unified}} = 75\% \times (1+1.99\% \times 50)+25\% \times (1+1+1.99\% \times 50) = 2.24 \]

How to Improve Cache Performance?

\[ \text{AMAT} = \text{Hit Time} + \text{Miss Rate} \times \text{Miss Penalty} \]

1. **Reduce the miss rate**.
2. Reduce the miss penalty, or
3. Reduce the time to hit in the cache.
Where do misses come from?

♦ Classifying Misses: 3 Cs

• **Compulsory**—The first access to a block is not in the cache, so the block must be brought into the cache. Also called **cold start misses** or **first reference misses** *(Misses in even an Infinite Cache)*

• **Capacity**—If the cache cannot contain all the blocks needed during execution of a program, **capacity misses** will occur due to blocks being discarded and later retrieved *(Misses in Fully Associative Size X Cache)*

• **Conflict**—If block-placement strategy is set associative or direct mapped, conflict misses (in addition to compulsory & capacity misses) will occur because a block can be discarded and later retrieved if too many blocks map to its set. Also called **collision misses** or **interference misses** *(Misses in N-way Associative, Size X Cache)*

♦ 4th “C”:

• **Coherence**—Misses caused by cache coherence

---

3Cs Miss Rate (SPEC92) vs. Cache size

♦ Old rule of thumb: 2x size => 25% cut in miss rate
Huge Caches - Working Sets

Example: LU Decomposition from NAS Parallel Benchmarks

Cache Organization impact

- Assume total cache size not changed
- Which of 3Cs is obviously affected?
- Change Block Size (only):

<table>
<thead>
<tr>
<th>Block Size (bytes)</th>
<th>Miss Rate</th>
</tr>
</thead>
<tbody>
<tr>
<td>16</td>
<td>0%</td>
</tr>
<tr>
<td>32</td>
<td>5%</td>
</tr>
<tr>
<td>64</td>
<td>10%</td>
</tr>
<tr>
<td>128</td>
<td>15%</td>
</tr>
<tr>
<td>256</td>
<td>20%</td>
</tr>
<tr>
<td>512</td>
<td>25%</td>
</tr>
</tbody>
</table>

Reduced compulsory misses

Increased Conflict Misses
Change Associativity: Conflict misses reduce with higher associativity but will AMAT go down?

- Example: ClockCycleTime CCT = 1.10 for 2-way, 1.12 for 4-way, 1.14 for 8-way vs. CCT direct-mapped

<table>
<thead>
<tr>
<th>Cache Size (KB)</th>
<th>1-way</th>
<th>2-way</th>
<th>4-way</th>
<th>8-way</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2.33</td>
<td>2.15</td>
<td>2.07</td>
<td>2.01</td>
</tr>
<tr>
<td>2</td>
<td>1.98</td>
<td>1.86</td>
<td>1.76</td>
<td>1.68</td>
</tr>
<tr>
<td>4</td>
<td>1.72</td>
<td>1.67</td>
<td>1.61</td>
<td>1.53</td>
</tr>
<tr>
<td>8</td>
<td>1.46</td>
<td>1.48</td>
<td>1.47</td>
<td>1.43</td>
</tr>
<tr>
<td>16</td>
<td>1.29</td>
<td>1.32</td>
<td>1.32</td>
<td>1.32</td>
</tr>
<tr>
<td>32</td>
<td>1.20</td>
<td>1.24</td>
<td>1.25</td>
<td>1.27</td>
</tr>
<tr>
<td>64</td>
<td>1.14</td>
<td>1.20</td>
<td>1.21</td>
<td>1.23</td>
</tr>
<tr>
<td>128</td>
<td>1.10</td>
<td>1.17</td>
<td>1.18</td>
<td>1.20</td>
</tr>
</tbody>
</table>

(Red means A.M.A.T. not improved by higher associativity)

Fast Hit Time + Low Conflict => Victim Cache

- How to combine fast hit time of direct mapped yet still avoid conflict misses?
- Add buffer to place data discarded from cache
- 4-entry victim cache removed 20% to 95% of conflicts for a 4 KB direct mapped data cache
- Used in Alpha, HP machines
Reducing Misses via "Pseudo-Associativity"

- How to combine fast hit time of Direct Mapped and have the lower conflict misses of 2-way SA cache?
- Divide cache: on a miss, check other half of cache to see if there, if so have a pseudo-hit (slow hit)

<table>
<thead>
<tr>
<th>Hit Time</th>
<th>Pseudo Hit Time</th>
<th>Miss Penalty</th>
</tr>
</thead>
</table>

- Drawback: CPU pipeline is complicated if hit takes 1 or 2 cycles
  - Better for caches not tied directly to processor (i.e., L2)
  - Used in MIPS R1000 L2 cache, similar in UltraSPARC

Reducing Misses by Hardware Prefetching of Instructions & Data

- Instruction Prefetching
  - Alpha 21064 fetches 2 blocks on a miss
  - Extra block placed in "stream buffer" - check upon miss
  - If hit in stream buffer, move stream buffer block into cache and prefetch next block (i+2)
- Works with data blocks too:
  - 1 data stream buffer reduced by 25% misses from 4KB cache; 4 streams reduced by 43%; 8 streams reduced by 50% to 70% misses from 64KB, 4-way set associative cache
Hardware Prefetching

♦ Strided prefetch
  • If observe sequence of accesses to block b, b+N, b+2N, then prefetch b+3N etc.
  • IBM Power 5 [2003] supports eight independent streams of strided prefetch per processor, prefetching 12 lines ahead of current access

♦ Intel processors use
  • Adjacent cache line prefetch from L2 to L1
  • HW prefetchers: instructions from L2 to L1 based on Branch prediction unit (BTB); Data and instruction from memory to L2 - triggered by successive cache misses and detection of a stride in accesses

♦ Prefetching relies on having extra memory bandwidth
  • Intel processors allow disabling of HW prefetch

Reducing Misses by **Software Prefetching Data**

- **Data Prefetching comes in two flavors:**
  - Load data into register (HP PA-RISC loads)
    - Binding prefetch: Must be correct address and register
  - Cache Prefetch: load into cache (MIPS IV, PowerPC, SPARC v. 9)
    - Non-Binding prefetch: Can be incorrect

```c
for(i=0; i < N; i++) {
    prefetch( &a[i + 1] );
    prefetch( &b[i + 1] );
    SUM = SUM + a[i] * b[i];
}
```

- Special prefetching instructions cannot cause faults; a form of speculative execution
- Issuing Prefetch Instructions takes time
  - Is cost of issuing prefetch < savings in reduced misses?
  - Higher superscalar reduces difficulty of issue bandwidth

---

Reducing Misses by **Compiler Optimizations**

- Cache misses reduced by 75% on 8KB direct mapped cache using software
- Instructions
  - Reorder procedures in memory so as to reduce conflict misses
  - Through profiling to detect conflicts
- Data
  - **Merging Arrays**: improve spatial locality by single array of compound elements vs. 2 arrays
  - **Loop Interchange**: change nesting of loops to access data in order stored in memory
  - **Loop Fusion**: Combine 2 independent loops that have same looping and some variables overlap
  - **Blocking**: Improve temporal locality by accessing “blocks” of data repeatedly vs. going down whole columns or rows
Merging Arrays Example

/* Before: 2 sequential arrays */
int val[SIZE];
int key[SIZE];

/* After: 1 array */
struct merge {
  int val;
  int key;
};

Reduce conflicts between val & key; improve spatial locality

Loop Interchange Example

/* Before */
for (j=0; j<100; j = j+1)
  for (i=0; i<5000; i = i+1)
    x[i][j] = 2 * x[i][j];

/* After */
for (i=0; i<5000; i = i+1)
  for (j=0; j<100; j = j+1)
    x[i][j] = 2 * x[i][j];

Sequential accesses instead of striding through memory every 100 words; improved spatial locality

Loop Fusion Example

for(i=0; i < N; i++)
  a[i] = b[i] * c[i];

for(i=0; i < N; i++)
  d[i] = a[i] * c[i];

for(i=0; i < N; i++)
  {
    a[i] = b[i] * c[i];
    d[i] = a[i] * c[i];
  }

Improve spatial locality
Summary of Compiler Optimizations to Reduce Cache Misses

<table>
<thead>
<tr>
<th>Function</th>
<th>Merged Arrays</th>
<th>Loop Interchange</th>
<th>Loop Fusion</th>
</tr>
</thead>
<tbody>
<tr>
<td>vpenta (nasa7)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>gmy (nasa7)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>tomcatv</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>btrix (nasa7)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>spice (nasa7)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>cholesky (nasa7)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>compress</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Reducing Miss Penalty:
(1) Read Priority over Write on Miss

- Write-through w/ write buffers => RAW hazards with main memory reads on cache misses
  - If simply wait for write buffer to empty, might increase read miss penalty (MIPS 1000 by 50%)
  - Check write buffer contents before read; if no conflicts, let the memory access continue
- Write-back: add buffer to hold displaced blocks
  - Read miss replacing dirty block
  - Normal: Write dirty block to memory, and then do the read
  - Instead copy the dirty block to a write buffer, then do the read, and then do the write
  - CPU stall less since restarts as soon as do read
Reduce Miss Penalty: (2) Early Restart and Critical Word First

- Don't wait for full block to be loaded before restarting CPU
  - **Early restart** — As soon as the requested word of the block arrives, send it to the CPU and let the CPU continue execution
  - **Critical Word First** — Request the missed word first from memory and send it to the CPU as soon as it arrives; let the CPU continue execution while filling the rest of the words in the block. Also called wrapped fetch and requested word first

- Generally useful only in large blocks
- Spatial locality ⇒ tend to want next sequential word, so not clear if benefit by early restart

---

Reduce Miss Penalty: (3) Non-blocking Caches to reduce stalls on misses

- **Non-blocking cache** or **lockup-free cache**: allow data cache to continue to supply cache hits during a miss
  - requires out-of-order execution
- “**hit under miss**” reduces the effective miss penalty by working during miss vs. ignoring CPU requests
- “**hit under multiple miss**” or “**miss under miss**” may further lower the effective miss penalty by overlapping multiple misses
  - Significantly increases the complexity of the cache controller as there can be multiple outstanding memory accesses
  - Requires multiple memory banks (otherwise cannot support)
  - Pentium Pro allows 4 outstanding memory misses
Benefits of Hit Under Miss for SPEC

- FP programs on average: AMAT = 0.68 -> 0.52 -> 0.34 -> 0.26
- Int programs on average: AMAT = 0.24 -> 0.20 -> 0.19 -> 0.19
- 8 KB Data Cache, Direct Mapped, 32B block, 16 cycle miss

(4): Add a second-level cache

- **L2 Equations**
  \[ AMAT = \text{Hit Time}_{L1} + \text{Miss Rate}_{L1} \times \text{Miss Penalty}_{L1} \]
  \[ \text{Miss Penalty}_{L1} = \text{Hit Time}_{L2} + \text{Miss Rate}_{L2} \times \text{Miss Penalty}_{L2} \]
  \[ AMAT = \text{Hit Time}_{L1} + \text{Miss Rate}_{L1} \times (\text{Hit Time}_{L2} + \text{Miss Rate}_{L2} \times \text{Miss Penalty}_{L2}) \]

- **Definitions:**
  - *Local miss rate*—misses in this cache divided by the total number of memory accesses to this cache (Miss rate\(_{L2}\))
  - *Global miss rate*—misses in this cache divided by the total number of memory accesses generated by the CPU
  - Global Miss Rate is what matters
L2 cache block size & AMAT

Relative to CPU Time

<table>
<thead>
<tr>
<th>Block Size</th>
<th>16</th>
<th>32</th>
<th>64</th>
<th>128</th>
<th>256</th>
<th>512</th>
</tr>
</thead>
<tbody>
<tr>
<td>AMAT</td>
<td>1.36</td>
<td>1.28</td>
<td>1.27</td>
<td>1.34</td>
<td>1.54</td>
<td>1.95</td>
</tr>
</tbody>
</table>

♦ 32KB L1, 8 byte path to memory

Cache Optimization Summary

<table>
<thead>
<tr>
<th>Technique</th>
<th>MRate</th>
<th>MP</th>
<th>HT</th>
<th>Complexity</th>
</tr>
</thead>
<tbody>
<tr>
<td>Larger Block Size</td>
<td>+</td>
<td>-</td>
<td>-</td>
<td>0</td>
</tr>
<tr>
<td>Higher Associativity</td>
<td>+</td>
<td>-</td>
<td>-</td>
<td>1</td>
</tr>
<tr>
<td>Victim Caches</td>
<td>+</td>
<td></td>
<td></td>
<td>2</td>
</tr>
<tr>
<td>Pseudo-Associative Caches</td>
<td>+</td>
<td></td>
<td></td>
<td>2</td>
</tr>
<tr>
<td>HW Prefetching of Instr/Data</td>
<td>+</td>
<td></td>
<td></td>
<td>3</td>
</tr>
<tr>
<td>Compiler Controlled Prefetching</td>
<td>+</td>
<td></td>
<td></td>
<td>3</td>
</tr>
<tr>
<td>Compiler Reduce Misses</td>
<td>+</td>
<td></td>
<td></td>
<td>0</td>
</tr>
</tbody>
</table>

| miss rate                              |       |    |    |            |
| Priority to Read Misses                | +     |    |    | 1          |
| Early Restart & Critical Word 1st      | +     |    |    | 2          |
| Non-Blocking Caches                    | +     |    |    | 3          |
| Second Level Caches                    | +     |    |    | 2          |

ECE568/Koren Part.13 .27 Adapted from UCB and other sources Copyright UCB & Morgan Kaufmann
Cache Misses vs. Time

- Cache misses are not distributed uniformly over time but tend to cluster
- Reload transients can be time consuming
- Each process (when run in isolation) has its own footprint in the cache

Cache Footprint of Process A

<table>
<thead>
<tr>
<th>Set 0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>A</td>
<td>A</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Set 1</td>
<td>A</td>
<td>A</td>
<td>A</td>
<td></td>
</tr>
<tr>
<td>Set 2</td>
<td>A</td>
<td>A</td>
<td>A</td>
<td>A</td>
</tr>
<tr>
<td>Set 3</td>
<td>A</td>
<td>A</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Set 4</td>
<td>A</td>
<td>A</td>
<td>A</td>
<td>A</td>
</tr>
<tr>
<td>Set 5</td>
<td>A</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Set 6</td>
<td>A</td>
<td>A</td>
<td>A</td>
<td></td>
</tr>
<tr>
<td>Set 7</td>
<td>A</td>
<td>A</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- Four-way set associative, 8 sets
- Size of A's footprint =
Footprint Distribution for k-way S.A. Cache

- Find the probability of having more than k blocks per set (total of N sets)
- Assume blocks distributed uniformly to sets
  - Prob (a block referenced by A falls into a given set) = p = 1/N
  - SA is the size of A's footprint
  - Use Binomial model with p and (1-p)
  - Prob (exactly i blocks of A in a single set) =

\[
\text{Prob} \{ X=i \} = \binom{SA}{i} p^i (1-p)^{(SA - i)} \quad \text{for } i < k
\]

\[
\sum_{j=k}^{SA} \binom{SA}{j} p^j (1-p)^{(SA - j)} \quad \text{for } i = k
\]

Similar distribution for process B with footprint SB

A & B compete for the cache

A then B

<table>
<thead>
<tr>
<th>Set 0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
</tr>
</thead>
<tbody>
<tr>
<td>B</td>
<td>B</td>
<td>A</td>
<td>A</td>
<td>A</td>
</tr>
<tr>
<td>Set 1</td>
<td>B</td>
<td>B</td>
<td>B</td>
<td>A</td>
</tr>
<tr>
<td>Set 2</td>
<td>B</td>
<td>A</td>
<td>A</td>
<td>A</td>
</tr>
<tr>
<td>Set 3</td>
<td>B</td>
<td>B</td>
<td>B</td>
<td>A</td>
</tr>
<tr>
<td>Set 4</td>
<td>B</td>
<td>A</td>
<td>A</td>
<td>A</td>
</tr>
<tr>
<td>Set 5</td>
<td>B</td>
<td>B</td>
<td>B</td>
<td>A</td>
</tr>
<tr>
<td>Set 6</td>
<td>B</td>
<td>B</td>
<td>A</td>
<td>A</td>
</tr>
<tr>
<td>Set 7</td>
<td>B</td>
<td>B</td>
<td>B</td>
<td>B</td>
</tr>
</tbody>
</table>

B then A

<table>
<thead>
<tr>
<th>Set 0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>A</td>
<td>A</td>
<td>B</td>
<td>B</td>
</tr>
<tr>
<td>Set 1</td>
<td>A</td>
<td>A</td>
<td>A</td>
<td>A</td>
</tr>
<tr>
<td>Set 2</td>
<td>A</td>
<td>A</td>
<td>A</td>
<td>A</td>
</tr>
<tr>
<td>Set 3</td>
<td>A</td>
<td>A</td>
<td>B</td>
<td>B</td>
</tr>
<tr>
<td>Set 4</td>
<td>A</td>
<td>A</td>
<td>A</td>
<td>A</td>
</tr>
<tr>
<td>Set 5</td>
<td>A</td>
<td>B</td>
<td>B</td>
<td></td>
</tr>
<tr>
<td>Set 6</td>
<td>A</td>
<td>A</td>
<td>B</td>
<td>B</td>
</tr>
<tr>
<td>Set 7</td>
<td>A</td>
<td>A</td>
<td>B</td>
<td>B</td>
</tr>
</tbody>
</table>

Most recently used items appear on the left
Size of Reload Transient for Set 1

- X (Y) denotes # of blocks in A's (B's) footprint in Set 1
- Z=X+Y total number of blocks in set 1
- If Z \leq k set 1 contributes nothing to reload transient:
  - Prob \{ Z \leq k \} = \sum_{i=0}^{k} ( \sum_{j=0}^{k-i} \text{Prob}\{X=i\} \text{Prob}\{Y=j\} )
  - X=i; Y=0,1,…, k-i
  - If X and Y Binomially distributed (p=1/N), Z=X+Y is too; this is the prob. of having k or less among \( S_A + S_B \)

A reload transient occurs when Z > k; let W blocks of A be overwritten by B

- Prob \{ W=i \} = \sum_{j=i}^{k} \text{Prob}\{X=j\} \text{Prob}\{Y=k+i-j\}; 1 \leq i \leq k

Cache Reload Transient

- Transient to reload A is: \( S_A - \{ \# \text{ of A's blocks that remain in cache when A resumes} \} \)

where \( E(X) = \sum_{i=0}^{k} i \text{Prob}\{X=i\} \)

\( S_A = 1900 \)
\( S_B = 7900 \)