Lecture 16: Memory Hierarchy—Misses, 3 Cs and 7 Ways to Reduce Misses

Professor Randy H. Katz
Computer Science 252
Spring 1996
Review: Who Cares About the Memory Hierarchy?

- Processor Only Thus Far in Course:
  - CPU cost/performance, ISA, Pipelined Execution

- 1980: no cache in µproc; 1995 2-level cache, 60% trans. on Alpha 21164 µproc (150 clock cycles for a miss!)
Review: Four Questions for Memory Hierarchy Designers

• Q1: Where can a block be placed in the upper level? (Block placement)
  – Fully Associative, Set Associative, Direct Mapped

• Q2: How is a block found if it is in the upper level? (Block identification)
  – Tag/Block

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

• Q4: What happens on a write? (Write strategy)
  – Write Back or Write Through (with Write Buffer)
**Review: Cache Performance**

CPU time = (CPU execution clock cycles + Memory stall clock cycles) x clock cycle time

Memory stall clock cycles = (Reads x Read miss rate x Read miss penalty + Writes x Write miss rate x Write miss penalty)

Memory stall clock cycles = Memory accesses x Miss rate x Miss penalty
Review: Cache Performance

CPUtime = IC \times (CPI_{\text{execution}} + \text{Mem accesses per instruction} \times \text{Miss rate} \times \text{Miss penalty}) \times \text{Clock cycle time}

\text{Misses per instruction} = \text{Memory accesses per instruction} \times \text{Miss rate}

\text{CPUtime} = \text{IC} \times (\text{CPI}_{\text{execution}} + \text{Misses per instruction} \times \text{Miss penalty}) \times \text{Clock cycle time}
Review: Improving Cache Performance

1. *Reduce the miss rate*,
2. Reduce the miss penalty, or
3. Reduce the time to hit in the cache.
Reducing Misses

- **Compulsory**—The first access to a block is not in the cache, so the block must be brought into the cache. These are also called *cold start misses* or *first reference misses.* *(Misses in 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 Size X Cache)*

- **Conflict**—If the block-placement strategy is set associative or direct mapped, conflict misses (in addition to compulsory and capacity misses) will occur because a block can be discarded and later retrieved if too many blocks map to its set. These are also called *collision misses* or *interference misses.* *(Misses in N-way Associative, Size X Cache)*
3Cs Absolute Miss Rate
3Cs Relative Miss Rate

Graph showing the relationship between cache size and miss rate per type (1-way, 2-way, 4-way, 8-way) for compulsory and conflict misses.
How Can Reduce Misses?

- Change Block Size? Which of 3Cs affected?
- Change Associativity? Which of 3Cs affected?
- Change Compiler? Which of 3Cs affected?
1. Reduce Misses via Larger Block Size

- Miss Rate (%):
  - 0%
  - 5%
  - 10%
  - 15%
  - 20%
  - 25%

- Block Size (bytes):
  - 16
  - 32
  - 64
  - 128
  - 256
  - 1K
  - 4K
  - 16K
  - 64K
  - 256K
2. Reduce Misses via Higher Associativity

• 2:1 Cache Rule:
  – Miss Rate DM cache size $N \approx$ Miss Rate FA cache size $N/2$

• Beware: Execution time is only final measure!
  – Will Clock Cycle time increase?
  – Hill [1988] suggested hit time external cache +10%, internal + 2% for 2-way vs. 1-way
Example: Avg. Memory Access Time vs. Miss Rate

- Example: assume 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 more associativity)
3. Reducing Misses via Victim Cache

- How to combine fast hit time of Direct Mapped yet still avoid conflict misses?
- Add buffer to place data discarded from cache
- Jouppi [1990]: 4-entry victim cache removed 20% to 95% of conflicts for a 4 KB direct mapped data cache
4. 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)
  
  ![Diagram showing hit time, pseudo-hit time, and miss penalty over time]

- Drawback: CPU pipeline is hard if hit takes 1 or 2 cycles
  - Better for caches not tied directly to processor
5. Reducing Misses by HW Prefetching of Instruction & Data

• E.g., Instruction Prefetching
  – Alpha 21064 fetches 2 blocks on a miss
  – Extra block placed in stream buffer
  – On miss check stream buffer

• Works with data blocks too:
  – Jouppi [1990] 1 data stream buffer got 25% misses from 4KB cache; 4 streams got 43%
  – Palacharla & Kessler [1994] for scientific programs for 8 streams got 50% to 70% of misses from 2 64KB, 4-way set associative caches

• Prefetching relies on extra memory bandwidth that can be used without penalty
6. Reducing Misses by SW Prefetching Data

• Data Prefetch
  – Load data into register (HP PA-RISC loads)
  – Cache Prefetch: load into cache (MIPS IV, PowerPC, SPARC v. 9)
  – Special prefetching instructions cannot cause faults; a form of speculative execution

• Issuing Prefetch Instructions takes time
  – Is cost of prefetch issues < savings in reduced misses?
7. Reducing Misses by Compiler Optimizations

• Instructions
  – Reorder procedures in memory so as to reduce misses
  – Profiling to look at conflicts
  – McFarling [1989] reduced caches misses by 75% on 8KB direct mapped cache with 4 byte blocks

• 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 */
int val[SIZE];
int key[SIZE];

/* After */
struct merge {
    int val;
    int key;
};
struct merge merged_array[SIZE];

Reducing conflicts between val & key
Loop Interchange Example

/* Before */
for (k = 0; k < 100; k = k+1)
    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 (k = 0; k < 100; k = k+1)
    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
Loop Fusion Example

/* Before */
for (i = 0; i < N; i = i+1)
  for (j = 0; j < N; j = j+1)
    a[i][j] = 1/b[i][j] * c[i][j];
for (i = 0; i < N; i = i+1)
  for (j = 0; j < N; j = j+1)
    d[i][j] = a[i][j] + c[i][j];
/* After */
for (i = 0; i < N; i = i+1)
  for (j = 0; j < N; j = j+1)
    { a[i][j] = 1/b[i][j] * c[i][j];
      d[i][j] = a[i][j] + c[i][j];}

2 misses per access to a & c vs. one miss per access
### Blocking Example

/* Before */

```c
for (i = 0; i < N; i = i+1)
    for (j = 0; j < N; j = j+1)
        {r = 0;
         for (k = 0; k < N; k = k+1) {
            r = r + y[i][k]*z[k][j];
         }
        x[i][j] = r;
    }
```

- **Two Inner Loops:**
  - Read all NxN elements of z[]
  - Read N elements of 1 row of y[] repeatedly
  - Write N elements of 1 row of x[]

- **Capacity Misses a function of N & Cache Size:**
  - 3 NxN => no capacity misses; otherwise ...

- **Idea:** compute on BxB submatrix that fits
/* After */
for (jj = 0; jj < N; jj = jj+B)
for (kk = 0; kk < N; kk = kk+B)
for (i = 0; i < N; i = i+1)
    for (j = jj; j < min(jj+B-1,N); j = j+1)
        {r = 0;
         for (k = kk; k < min(kk+B-1,N); k = k+1) {
            r = r + y[i][k]*z[k][j];};
         x[i][j] = x[i][j] + r;}

• Capacity Misses from $2N^3 + N^2$ to $2N^3/B + N^2$
• B called **Blocking Factor**
• Conflict Misses Too?
Reducing Conflict Misses by Blocking

• Conflict misses in caches not FA vs. Blocking size
  – Lam et al [1991] a blocking factor of 24 had a fifth the misses vs. 48 despite both fit in cache
Summary of Compiler Optimizations to Reduce Cache Misses

- **merged arrays**
- **loop interchange**
- **loop fusion**
- **blocking**

<table>
<thead>
<tr>
<th>Function</th>
<th>Performance Improvement</th>
</tr>
</thead>
<tbody>
<tr>
<td>vpenta (nasa7)</td>
<td>2.5</td>
</tr>
<tr>
<td>gmtty (nasa7)</td>
<td>2.6</td>
</tr>
<tr>
<td>tomcatv</td>
<td>1.5</td>
</tr>
<tr>
<td>btrix (nasa7)</td>
<td>1.3</td>
</tr>
<tr>
<td>mxm (nasa7)</td>
<td>1.3</td>
</tr>
<tr>
<td>spice cholesky (nasa7)</td>
<td>1.3</td>
</tr>
<tr>
<td>compress</td>
<td>1.3</td>
</tr>
</tbody>
</table>
Summary

$$CPUtil = IC \times \left( CPI_{\text{Execution}} + \frac{\text{Memory accesses}}{\text{Instruction}} \times \text{Miss rate} \times \text{Miss penalty} \right) \times \text{Clock cycle time}$$

- **3 Cs: Compulsory, Capacity, Conflict Misses**
- **Reducing Miss Rate**
  1. Reduce Misses via Larger Block Size
  2. Reduce Misses via Higher Associativity
  3. Reducing Misses via Victim Cache
  4. Reducing Misses via Pseudo-Associativity
  5. Reducing Misses by HW Prefetching Instr, Data
  6. Reducing Misses by SW Prefetching Data
  7. Reducing Misses by Compiler Optimizations
- **Remember danger of concentrating on just one parameter when evaluating performance**
- **Next lecture: reducing Miss penalty**