CMPSC 311 - Introduction to Systems Programming
Module: Caching

Professor Patrick McDaniel
Fall 2015
• Midterm moved to November 4th (next Wed)
  ‣ Both caching (today) and signals (Mon) on test
• Superfluous output (HERE) on assign3, grab new libraid.a downloadable from website
• Assignment submission
  ‣ Some students had many, many problems sending me email
  ‣ New rule: whatever is in my inbox submission, nothing else matters (you are to be responsible), tips:
    • Do a “make clean” before tarring up the file
    • CC yourself, make sure you got it
int show_cache(void) {

    // Local variables
    int memory[MAX_VALUE], idx, i;
    memset(memory, 0x0, sizeof(memory));

    // Set the values of the array
    for (i=0; i<10; i++) {
        idx = random()%MAX_VALUE;
        memory[idx] = i;
        printf("time %d, memory[%d] = %d\n", i, idx, i);
    }

    return(0);
}
Sample program

```c
int show_cache(void) {
    // Local variables
    int memory[MAX_VALUE], idx, i;
    memset(memory, 0x0, sizeof(memory));

    // Set the values of the array
    for (i=0; i<10; i++) {
        idx = random()%MAX_VALUE;
        memory[idx] = i;
        printf("time %d, memory[%d] = %d\n", i, idx, i);
    }
    return(0);
}
```

Memory accesses: 1, 4, 3, 1, 5, 1, 4, 0, 3, 1
Reminder: Memory Hierarchy

- **L0:** Registers
  - CPU registers hold words retrieved from L1 cache

- **L1:** L1 cache (SRAM)
  - L1 cache holds cache lines retrieved from L2 cache

- **L2:** L2 cache (SRAM)
  - L2 cache holds cache lines retrieved from main memory

- **L3:** Main memory (DRAM)
  - Main memory holds disk blocks retrieved from local disks

- **L4:** Local secondary storage (local disks)
  - Local disks hold files retrieved from disks on remote network servers

- **L5:** Remote secondary storage (tapes, distributed file systems, Web servers)

---

Smaller, faster, costlier per byte

Larger, slower, cheaper per byte

Processor Caches

• Most modern computers have multiple layers of caches to manage data passing into and out of the processors
  ‣ L1 – very fast and small, processor adjacent
  ‣ L2 – a bit slower but often much larger
  ‣ L3 – larger still, maybe off chip
    • May be shared amongst processors in multi-core system
  ‣ Memory – slowest, least expensive
• Instruction caches are different from data caches
Caches

- **Cache**: A smaller, faster storage device that acts as a staging area for a subset of the data in a larger, slower device.

- Fundamental idea of a memory hierarchy:
  - For each k, the faster, smaller device at level k serves as a cache for the larger, slower device at level k+1.

- Why do memory hierarchies work?
  - Because of locality, programs tend to access the data at level k more often than they access the data at level k+1.
  - Thus, the storage at level k+1 can be slower, and thus larger and cheaper per bit.

- **Big Idea**: The memory hierarchy creates a large pool of storage that costs as much as the cheap storage near the bottom, but that serves data to programs at the rate of the fast storage near the top.
Locality

• Caches exploit locality to improve performance, of which there are two types:
  ‣ **Spatial locality**: accessed data used is tend to be close to data you already accessed
  ‣ **Temporal (time) locality**: data that is accessed is likely to be accessed again soon

• This leads to two cache design strategies
  ‣ Spatial: cache items in blocks larger than that accessed
  ‣ Temporal: keep stuff used recently around longer
General Cache Concepts

Cache

8  9  14  3
General Cache Concepts

Cache

Cache Lines

8 9 14 3
### General Cache Concepts

<table>
<thead>
<tr>
<th>Cache</th>
<th>Memory</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td>0</td>
</tr>
<tr>
<td>9</td>
<td>1</td>
</tr>
<tr>
<td>10</td>
<td>2</td>
</tr>
<tr>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td></td>
<td>4</td>
</tr>
<tr>
<td></td>
<td>5</td>
</tr>
<tr>
<td></td>
<td>6</td>
</tr>
<tr>
<td></td>
<td>7</td>
</tr>
<tr>
<td>4</td>
<td>8</td>
</tr>
<tr>
<td>5</td>
<td>9</td>
</tr>
<tr>
<td>6</td>
<td>10</td>
</tr>
<tr>
<td>7</td>
<td>11</td>
</tr>
<tr>
<td>8</td>
<td>12</td>
</tr>
<tr>
<td>9</td>
<td>13</td>
</tr>
<tr>
<td>10</td>
<td>14</td>
</tr>
<tr>
<td>11</td>
<td>15</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- Smaller, faster, more expensive memory caches a subset of the blocks.
- Data is copied in block-sized transfer units.
- Larger, slower, cheaper memory viewed as partitioned into “blocks.”
Cache Hit

Data in block b is needed

Block b is in cache:
Hit!
Cache Miss

Data in block b is needed

Block b is not in cache: Miss!

Block b is fetched from memory

Block b is stored in cache

• Placement policy: determines where b goes
• Replacement policy: determines which block gets evicted (victim)
Placement Policy

• Q: When a new block comes in, where in the cache can you keep it?
• A: Depends on the placement policy
  ‣ Anywhere (fully associative)
    • Why not do this all the time?
  ‣ Exactly one cache line (direct-mapped)
    • Commonly, block i is mapped to cache line (i mod t) where t is the total number of lines
  ‣ One of n cache lines (n-way set-associative)
Placement Policy

- Q: When a new block comes in, where in the cache can you keep it?
- A: Depends on the placement policy
  - Anywhere (fully associative)
    - Why not do this all the time?
  - Exactly one cache line (direct-mapped)
    - Commonly, block i is mapped to cache line (i mod t) where t is the total number of lines
  - One of n cache lines (n-way set-associative)
Placement Policy

• Q: When a new block comes in, where in the cache can you keep it?
• A: Depends on the placement policy
  ‣ Anywhere (fully associative)
    • Why not do this all the time?
  ‣ Exactly one cache line (direct-mapped)
    • Commonly, block \( i \) is mapped to cache line \( (i \mod t) \) where \( t \) is the total number of lines
  ‣ One of \( n \) cache lines (\( n \)-way set-associative)
Types of Cache Misses

• Cold (compulsory) miss
  ▸ Cold misses occur because the cache is empty.

• Capacity miss
  ▸ Occurs when the set of active cache blocks (working set) is larger than the cache.

• Conflict miss (direct mapping only)
  ▸ Most caches limit blocks at level k+1 to a small subset (sometimes a singleton) of the block positions at level k.
    • E.g. Block i at level k+1 must be placed in block (i mod 4) at level k.
  ▸ Conflict misses occur when the level k cache is large enough, but multiple data objects all map to the same level k block.
    • E.g. Referencing blocks 0, 8, 0, 8, 0, 8, ... would miss every time.
Conflict Miss

![Diagram of Level K+1 and Level K with keys X, Y, A, B, G, K]
Conflict Miss

![Diagram showing Level K+1 and Level K with a conflict miss.]
Conflict Miss
Conflict Miss

![Diagram showing Level K+1 and Level K with conflicts marked]
Cache replacement policy

- When your cache is full and you acquire a new value, you must evict a previously stored value
  - Performance of cache is determined by how smart you are in evicting values, known as a *cache eviction policy*
  - Popular policies
    - *Least recently used* (LRU) - eject the value that has been in the cache the longest without being accessed
    - *Least frequently used* (LFU) - eject the value that accessed the least number of times
    - *First in-first out* (FIFO) - eject the same order they come in
  - Policy efficiency is measured by the hit performance (how often is something asked for and found) and measured costs
    - Determined by *working set* and *workload*
Cache performance

- A cache hit is when the referenced information is served out of the cache.
- A cache miss occurs when referenced information cannot be served out of the cache.
- The hit ratio is the:

  $\text{hit ratio} = \frac{\# \text{ cache hits}}{\# \text{ total accesses}}$

- Note that the efficiency of a cache is almost entirely determined by the hit ratio.
Cache performance

• The *average memory access time* can be calculated:

\[
\text{memory latency} = \text{hit cost} + P(\text{miss}) \times \text{miss penalty}
\]

• Where

  ‣ hit cost is the cost to access out of cache
  ‣ miss penalty is the cost to serve out of main memory
  ‣ \( P(\text{miss}) \) is the probability of a cache access resulting in a miss, e.g., ratio of hits/total accesses

• E.g., for a hit cost of 25 usec and, penalty of 250 usec, and cache hit rate of 80%:

\[
25 \text{ usec} + (0.2 \times 250 \text{ usec}) = 25 + 50 \text{ usec} = 75 \text{ usec}
\]

• This is the average access time through the cache.
Example: 4 Line LRU Cache

<table>
<thead>
<tr>
<th>Mem</th>
<th>Time</th>
<th>Mem</th>
<th>Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td></td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>T=0</td>
<td></td>
<td>T=0</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td></td>
<td>4</td>
<td>1</td>
</tr>
<tr>
<td>T=1</td>
<td></td>
<td>T=1</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>1</td>
<td>4</td>
<td></td>
</tr>
<tr>
<td>T=2</td>
<td>0</td>
<td>T=0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>4</td>
<td>3</td>
</tr>
<tr>
<td>T=3</td>
<td>0</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>5</td>
<td>3</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>T=4</td>
<td></td>
<td>T=3</td>
<td></td>
</tr>
</tbody>
</table>

(time 0, memory[1] = 0)
(time 1, memory[4] = 1)
(time 2, memory[3] = 2)
(time 3, memory[1] = 3)
(time 4, memory[5] = 4)
(time 5, memory[1] = 5)
(time 6, memory[4] = 6)
(time 7, memory[0] = 7)
(time 8, memory[3] = 8)
(time 9, memory[1] = 9)
Example: 4 Line LRU Cache

<table>
<thead>
<tr>
<th>T=5</th>
<th>Mem</th>
<th>1</th>
<th>4</th>
<th>3</th>
<th>5</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Time</td>
<td>3</td>
<td>1</td>
<td>2</td>
<td>4</td>
</tr>
</tbody>
</table>

(hit)

<table>
<thead>
<tr>
<th>T=6</th>
<th>Mem</th>
<th>1</th>
<th>4</th>
<th>3</th>
<th>5</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Time</td>
<td>5</td>
<td>1</td>
<td>2</td>
<td>4</td>
</tr>
</tbody>
</table>

(hit)

<table>
<thead>
<tr>
<th>T=7</th>
<th>Mem</th>
<th>1</th>
<th>4</th>
<th>3</th>
<th>5</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Time</td>
<td>5</td>
<td>6</td>
<td>2</td>
<td>4</td>
</tr>
</tbody>
</table>

(miss)

<table>
<thead>
<tr>
<th>T=8</th>
<th>Mem</th>
<th>1</th>
<th>4</th>
<th>0</th>
<th>5</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Time</td>
<td>5</td>
<td>6</td>
<td>7</td>
<td>4</td>
</tr>
</tbody>
</table>

(miss)

<table>
<thead>
<tr>
<th>T=9</th>
<th>Mem</th>
<th>1</th>
<th>4</th>
<th>0</th>
<th>3</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Time</td>
<td>5</td>
<td>6</td>
<td>7</td>
<td>8</td>
</tr>
</tbody>
</table>

(hit)
Example: 4 Line LRU Cache

- Result: 6 misses, 4 hits
  - Pr(miss) = 0.6
- Assume
  - Hit cost (100 usec)
  - Miss penalty (1000 usec)

- So the average memory access time is:

  \[ 100 \text{ usec} + (0.6 \times 1000 \text{ usec}) = 100 + 600 = 700 \text{ usec} \]

- Q: Why is the performance so poor?