Caching and Virtual Memory

Last Time

- Flexible Address Translation
  - Segmentation + paged translation
  - Multi-level paged translation
  - Hashing

- Efficient Address Translation
  - Translation Lookaside Buffers (TLBs)
  - *Virtually addressed and physically addressed* caches

Main Points

- Cache concept
  - Hardware vs. software caches

- When caches work and when they don’t
  - Spatial/temporal locality vs. Zipf workloads

- Cache replacement policies
Multicore and Hyperthreading

- Modern CPU has several functional units
  - Instruction decode
  - Arithmetic/branch
  - Floating point
  - Instruction/data cache
  - TLB
- Multicore: replicate functional units (i7: 4)
  - Share second/third level cache, second level TLB
- Hyperthreading: logical processors that share functional units (i7: 2)
  - Better functional unit utilization during memory stalls
- No difference from the OS/programmer perspective
  - Except for performance, affinity, ...

Definitions

- Cache
  - Copy of data that is faster to access than the original
  - Hit: if cache has copy
  - Miss: if cache does not have copy
- Cache block
  - Unit of cache storage (multiple memory locations)
- Temporal locality
  - Programs tend to reference the same memory locations multiple times
  - Example: instructions in a loop
- Spatial locality
  - Programs tend to reference nearby locations
  - Example: data in a loop

Cache Concept (Read)

Cache

<table>
<thead>
<tr>
<th>fetch address</th>
<th>address in cache?</th>
<th>no</th>
<th>yes</th>
<th>fetch address</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Cache Concept (Write)

Cache

<table>
<thead>
<tr>
<th>store value at address</th>
<th>store value in cache?</th>
<th>no</th>
<th>yes</th>
<th>fetch address</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Write through: changes sent immediately to next level of storage
Write back: changes stored in cache until cache block is replaced
Memory Hierarchy

<table>
<thead>
<tr>
<th>Cache</th>
<th>Hit Cost</th>
<th>Size</th>
</tr>
</thead>
<tbody>
<tr>
<td>1st level cache/first level TLB</td>
<td>1 ns</td>
<td>64 KB</td>
</tr>
<tr>
<td>2nd level cache/second level TLB</td>
<td>4 ns</td>
<td>256 KB</td>
</tr>
<tr>
<td>3rd level cache</td>
<td>12 ns</td>
<td>2 MB</td>
</tr>
<tr>
<td>Memory (DRAM)</td>
<td>100 ns</td>
<td>10 GB</td>
</tr>
<tr>
<td>Data center memory (DRAM)</td>
<td>100 µs</td>
<td>100 TB</td>
</tr>
<tr>
<td>Local non-volatile memory</td>
<td>100 µs</td>
<td>100 GB</td>
</tr>
<tr>
<td>Local disk</td>
<td>10 ms</td>
<td>1 TB</td>
</tr>
<tr>
<td>Data center disk</td>
<td>10 ms</td>
<td>100 PB</td>
</tr>
<tr>
<td>Remote data center disk</td>
<td>200 ms</td>
<td>1 XB</td>
</tr>
</tbody>
</table>

i7 has 8MB as shared 3rd level cache; 2nd level cache is per-core

Hardware Design Principle

The bigger the memory, the slower the memory

Address Translation with TLB

Virtual Addressed Caches
Questions

• With a virtual cache, what do we need to do on a context switch?
• What if the virtual cache > page size?
  – Page size: 4KB (x86)
  – First level cache size: 64KB (i7)
  – Cache block size: 32 bytes

Aliasing

• Alias: two (or more) virtual cache entries that refer to the same physical memory
  – What if we modify one alias and then context switch?
• Typical solution
  – On a write, lookup virtual cache and TLB in parallel
  – Physical address from TLB used to check for aliases

Memory Hierarchy

<table>
<thead>
<tr>
<th>Cache</th>
<th>Hit Cost</th>
<th>Size</th>
</tr>
</thead>
<tbody>
<tr>
<td>1st level cache/first level TLB</td>
<td>1 ns</td>
<td>64 KB</td>
</tr>
<tr>
<td>2nd level cache/second level TLB</td>
<td>4 ns</td>
<td>256 KB</td>
</tr>
<tr>
<td>3rd level cache</td>
<td>12 ns</td>
<td>2 MB</td>
</tr>
<tr>
<td>Memory (DRAM)</td>
<td>100 ns</td>
<td>10 GB</td>
</tr>
<tr>
<td>Data center memory (DRAM)</td>
<td>100 μs</td>
<td>100 TB</td>
</tr>
<tr>
<td>Local non-volatile memory</td>
<td>100 μs</td>
<td>100 GB</td>
</tr>
<tr>
<td>Local disk</td>
<td>10 ns</td>
<td>1 TB</td>
</tr>
<tr>
<td>Data center disk</td>
<td>10 ms</td>
<td>100 PB</td>
</tr>
<tr>
<td>Remote data center disk</td>
<td>200 ms</td>
<td>1 ×B</td>
</tr>
</tbody>
</table>

i7 has 8MB as shared 3rd level cache; 2nd level cache is per-core

Translation on a Modern Processor
Question

• What is the cost of a first level TLB miss?
  – Second level TLB lookup
• What is the cost of a second level TLB miss?
  – x86: 2-4 level page table walk
• How expensive is a 4-level page table walk on a modern processor?

Working Set Model

• Working Set: set of memory locations that need to be cached for reasonable cache hit rate
• Thrashing: when system has too small a cache

Phase Change Behavior

• Programs can change their working set
• Context switches also change working set

Zipf Distribution

• Caching behavior of many systems are not well characterized by the working set model
• An alternative is the Zipf distribution
  – Popularity ~ 1/k^c, for kth most popular item, 1 < c < 2
Zipf Distribution

Zipf Examples
- Web pages
- Movies
- Library books
- Words in text
- Salaries
- City population
- ...

Common thread: popularity is self-reinforcing

Zipf and Caching

Cache Lookup: Fully Associative

address

value

match at any address?

yes

return value
Cache Lookup: Direct Mapped

<table>
<thead>
<tr>
<th>hash(address)</th>
<th>value</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- match at hash(address)?
  - yes
    - return value

Cache Lookup: Set Associative

<table>
<thead>
<tr>
<th>address</th>
<th>value</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x0053</td>
<td></td>
</tr>
</tbody>
</table>

- match at hash(address)?
  - yes
    - return value

<table>
<thead>
<tr>
<th>address</th>
<th>value</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x120d</td>
<td></td>
</tr>
</tbody>
</table>

- match at hash(address)?
  - yes
    - return value

Page Coloring

- What happens when cache size >> page size?
  - Direct mapped or set associative
  - Multiple pages map to the same cache line
- OS page assignment matters!
  - Example: 8MB cache, 4KB pages
  - 1 of every 2K pages lands in same place in cache
- What should the OS do?

Cache Replacement Policy

- On a cache miss, how do we choose which entry to replace?
  - Assuming the new entry is more likely to be used in the near future
  - In direct mapped caches, not an issue!
- Policy goal: reduce cache misses
  - Improve expected case performance
  - Also: reduce likelihood of very poor performance
A Simple Policy

- Random?
  - Replace a random entry

- FIFO?
  - Replace the entry that has been in the cache the longest time
  - What could go wrong?

FIFO in Action

Worst case for FIFO is if program strides through memory that is larger than the cache

MIN, LRU, LFU

- MIN
  - Replace the cache entry that will not be used for the longest time into the future
  - Optimality proof based on exchange: if evict an entry used sooner, that will trigger an earlier cache miss

- Least Recently Used (LRU)
  - Replace the cache entry that has not been used for the longest time in the past
  - Approximation of MIN

- Least Frequently Used (LFU)
  - Replace the cache entry used the least often (in the recent past)

LRU/MIN for Sequential Scan