EC 413
Computer Organization

Cache Structures

Prof. Michel A. Kinsy

Processor-Memory Gap

- Performance gap: CPU (55% each year) vs. DRAM (7% each year)
  - Processor operations take of the order of 1 ns
  - Memory access requires 10s or even 100s of ns
  - Each instruction executed involves at least one memory access

![Graph showing performance gap between processor and memory]

Computer Organization v5

- Our sixth view of computer organization
- A fast memory can help bridge the CPU-memory gap
- The fastest memories are expensive and thus not very large
Address Bit-Field Partitioning

- The address (e.g., 32-bit) issued by the CPU is generally divided into 3 fields
  - Tag
    - Serves as the unique identifier for a group of data
    - Different regions of memory may be mapped to the same cache location/block
    - The tag is used to differentiate between them
  - Index
    - It is used to index into the cache structure
  - Block Offset
    - The least significant bits are used to determine the exact data word
    - If the block size is B then b = \( \log_2 B \) bits will be needed in the address to specify data word

<table>
<thead>
<tr>
<th>Address</th>
<th>Tag</th>
<th>Index</th>
<th>Block Offset</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>t bits</td>
<td>k bits</td>
<td>b bits</td>
</tr>
</tbody>
</table>

Caching & Cache Structures

- cache is transparent to user (happens automatically)
Caches

- Local miss rate = misses in cache / accesses to cache
- Global miss rate = misses in cache / CPU memory accesses
- Misses per instruction = misses in cache / number of instructions

Caching principles

- Cache size (in bytes or words)
  - Total cache capacity
  - A larger cache can hold more of the program's useful data but is more costly and likely to be slower
- Block or cache-line size
  - Unit of data transfer between cache and main
  - With a larger cache line, more data is brought in cache with each miss. This can improve the hit rate but also may bring low-utility data in cache

Caching principles

- Placement policy
  - Determining where an incoming cache line is stored
  - More flexible policies imply higher hardware cost and may or may not have performance benefits (due to more complex data location)
- Replacement policy
  - Determining which of several existing cache blocks (into which a new cache line can be mapped) should be overwritten
  - Typical policies: choosing a random or the least recently used block
Caching Principles

- Compulsory misses
  - With on-demand fetching, first access to any item is a miss

- Capacity misses
  - We have to evict some items to make room for others
  - This leads to misses that are not incurred with an infinitely large cache

- Conflict misses
  - The placement scheme may force us to displace useful items to bring in other items
  - This may lead to misses in future

Line width ($2^W$)

- Too small a value for $W$ causes a lot of main memory accesses
- Too large a value increases the miss penalty and may tie up cache space with low-utility items that are replaced before being used

Set size or associativity ($2^S$)

- Direct mapping ($S = 0$) is simple and fast
- Greater associativity leads to more complexity, and thus slower access, but tends to reduce conflict misses

Cache Algorithm (Read)

- Look at Processor Address, search cache tags to find match. Then either

  - Found in cache a.k.a. HIT
    - Return copy of data from cache
  - Not in cache a.k.a. MISS
    - Read block of data from Main Memory
    - Wait ...
    - Return data to processor and update cache

Q: Which line do we replace?
Placement Policy

- There are multiple approaches
- Modulo operation creates a set that we can use for placing data blocks into the cache
- The result of the modulo operation with modulus $n$ is always an integer between 0 and $n-1$
Caching Mechanism

<table>
<thead>
<tr>
<th>Cache Indexes</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
</tr>
<tr>
<td>10</td>
</tr>
<tr>
<td>10</td>
</tr>
</tbody>
</table>

Address generated by the processor: 4, 8, 10, 15, 9, 7, 6

Memory:

| 00000 (00) | 00100 (04) | 01000 (08) | 11000 (24) |

Cache Indexes:

|  0  |  8  | 12  |
|  10 |  16 |  20 |
|  10 |  16 |  20 |

Address generated by the processor: 4, 8, 10, 15, 9, 7, 6

Memory:

| 00000 (00) | 00100 (04) | 01000 (08) | 11000 (24) |
Caching Principles

- Cache contains copies of some of Main Memory
  - Those storage locations recently used
  - When Main Memory address A is referenced in CPU
    - Cache checked for a copy of contents of A
      - If found, cache hit
        - Copy used
          - No need to access Main Memory
      - If not found, cache miss
        - Main Memory accessed to get contents of A
          - Copy of contents also loaded into cache

Next Class

- Cache Performance