CMPT 295
Unit – Memory Hierarchy
Lecture 35 – Caching
Last lecture

- Speed separation between registers (1 clock cycle / access) and main memory (60 clock cycles / access) is huge
- To narrow this gap, add cache
  - Use faster memory components (SRAM: 4 cycles / access) to hold copy of portion of main memory likely to be used in near future
  - This takes advantage of locality
    - Temporal locality: access the same location again soon
    - Spatial locality: access a nearby location soon
- Understanding memory hierarchy and caching allows us to write cache-friendly code
  - Programs with good locality tend to run faster because they maximizes data access from higher levels of memory hierarchy (i.e., from faster memory components)
Today’s Menu

- Locality – Cache-friendly code optimization
- Memory hierarchy
- Caching
- Cache Memories
How is cache structured?

- Memory model:
  - Size of memory: \( M = 2^m \times 8 \text{ bits} \)
  - Memory address: \( m \) bits
  - Memory resolution: 1 Byte
  - Cache \( C \) contains \( S \) sets = \( 2^s \) sets
    - Each set contains \( E \) cache lines
    - Each line contains:
      - 1 data block \( B = 2^b \) bytes
      - 1 valid bit \( v \)
      - \( t \) tag bits, where \( t = m - (b + s) \)
    - Defined as tuple \((S, E, B, m)\)
  - Capacity of \( C = S \times E \times B \)
How does cache work?

- When CPU executes a LOAD instruction
  - `LOAD a, rC` \(\rightarrow\) read data from memory at this particular address
  - CPU actually reads the content of memory address \(a\) from L1 cache instead of from main memory
  - If content of address \(a\) is in L1 cache
    - Cache hit and content of address \(a\) returned to CPU
  - If content of address \(a\) is not in L1 cache
    - Cache miss and content of address \(a\) is read from the next level down, i.e., main memory
    - If content of address \(a\) is in main memory
      - Cache hit and content of address \(a\) is copied to L1 cache and then returned to CPU
    - If content of address \(a\) is not in main memory
      - Cache miss and content of address \(a\) is read from the next level down ...

### Diagram

- CPU Registers
- L1 Cache
- Main Memory
Cache miss - Worst scenario

L0 - CPU Registers

L1 Cache

L2 Cache

L3 Cache

L4 - Main Memory

L5 - Local secondary storage (local disks)
But how does cache know?

How does the cache know it contains a copy of content of memory address \( a \) ?

**Memory address:**

1. \( s \) \rightarrow number of bits uniquely identifying a set (set index)
   - **Set index** tells us in which set to look for data
2. \( t \) \rightarrow number of tag bits
   - **Tag** tells us in which active line of a set “set index” to look for data
3. \( b \) \rightarrow number of bits defining the block offset
   - **Block offset** tells us where to find data within block B

Read as unsigned integers
Example: Capacity of cache defined as $(S, E, B, m) = (4, 2, 64, 48)$?

- Memory model:
  - Size of memory: $M = 2^m \times 8$ bits
  - Memory address: m bits
  - Memory resolution: 1 Byte
  - Cache C contains $S$ sets = $2^s$ sets
    - Each set contains $E$ cache lines
    - Each line contains:
      - 1 data block $B = 2^b$ bytes
      - 1 valid bit $v$
      - $t$ tag bits, where $t = m - (b + s)$
    - Defined as tuple $(S, E, B, m)$
    - Capacity of C = $S \times E \times B$

<table>
<thead>
<tr>
<th>v</th>
<th>Tag Bits</th>
<th>Block Bytes</th>
</tr>
</thead>
<tbody>
<tr>
<td>Set 0 {</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Set 1 {</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Set 2 {</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Set 3 {</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- $S = 4 \rightarrow s = 2$
- $E = 2$ lines per set
- $B = 64 \rightarrow b = 6$
- $m = 48$
- Capacity of C = $4 \times 2 \times 64 = 2^9$ = 512 bytes
Example: L1 -> (S, E, B, m) = (4, 2, 64, 48)
Assuming memory hierarchy: CPU register – L1 – main memory

1. Iteration 1 of sumArray1:
   C: sum += A[0][0];
   Assembly code: `addl (%rdi,%rcx,4),%eax`
   Micro instruction: `load 0x5EFD80`
   - CPU sends load request to L1
   
   0x0000005EFD80:
   1. Set Selection: 10₂ → 2₁₀
   2. Line Matching: Is v valid? 0 → invalid!
   → L1 cache miss

Since cache is empty (start of program execution), it is called a cold cache and this type of misses are called cold miss
Example: L1 -> \((S, E, B, m) = (4, 2, 64, 48)\)

Assuming memory hierarchy: CPU register – L1 – main memory

1. Iteration 1 of sumArray1:

   \(0x0000005EFD80\):

   - Content of \(0x0...05EFD80\) to \(0x0...05EFDBF\) \((\text{M}[0x0...05EFD80] \to \text{M}[0x0...05EFDBF]\)) read from main memory and copied into Set 2 of L1 -> expensive (takes time)

   - Tag \(t = 0x5EFD\) and valid \(v = 1\)

2. Word extraction: Cache returns content of block at offset 0 to CPU -> 4 bytes (int)
Example: \((S, E, B, m) = (4, 2, 64, 48)\)
Assuming memory hierarchy: CPU register – L1 – main memory

2. Iteration 6 of sumArray1:
   C: \(\text{sum } += \ A[1][1];\)
   Assembly code: \(\text{addl (}\%\text{rdi, }\%\text{rcx, 4}), \%\text{eax}\)
   Micro instruction: \(\text{load 0x5EFD94}\)
   ➤ CPU sends \(\text{load}\) request to L1

\[
0x00000005EFD94:\begin{array}{cccc}
0 & 0 & 1 & 0
\end{array}
\]

1. Set Selection: \(10_2 \rightarrow 2_{10}\)
2. Line Matching:
   ➤ Is \(v\) valid? \(1 \rightarrow \text{valid!}\)
   ➤ Is tag matching? Yes, then …
   ➤ L1 cache hit
3. Word extraction: Cache returns content of block at offset 20 to CPU \(\rightarrow 4\) bytes (int)
Eviction Strategy

If there are no empty lines in a set ...

1. *Random*: select line to be overwritten randomly
   - Fast and easy to implement in h/w

2. *Least recently used* (LRU): select “oldest” line as the line to be overwritten
   - Based on principle of locality
   - Need to keep track of stats → extra h/w

3. *Least frequently used* (LFU): select least referenced line over time $T$ as the line to be overwritten
   - Based on principle of locality
   - Need to keep track of stats → extra h/w
Types of cache - \((S, E, B, m)\)

1. Our example used a set associative cache

2. If \(E = 1\) → one line per set
   - Direct-mapped cache
   - Simple “line matching” process
   - Issue: conflict miss

3. If \(S = 1\) → one set
   - Fully-associative cache
   - No set index bits in memory address → no “set selection” set
   - Issue: Tag comparisons – done in parallel
     - Expensive to build
Performance Analysis of sumArray1

- Cache miss: 1
- Cache hit: 11
- Total # of accesses: 12
- What is the average memory access time?

average memory access time = hit rate \times hit time + miss rate \times miss time

\[
\frac{11}{12} \times 4 \text{ c.c.} + \frac{1}{12} \times 200 \text{ c.c.} = 4 \text{ c.c. (since L1 has access time of 4 clock cycles) + 200 c.c. (since main memory has access time of 200 clock cycles)}
\]
Figure 6.41 A memory mountain. Shows read throughput as a function of temporal and spatial locality.
Strides of 64 bytes and greater create lowest throughput because accessing the next element 64 bytes away signifies that the next element will not be found in the same cache line as the previous element -> cache miss

As stride increases, throughput decreases

Figure 6.43  A slope of spatial locality. The graph shows a slice through Figure 6.41 with size = 4 MB.
Ridges of memory mountain follow cache limits: 32 KB, 256 KB, 8 MB

As data size increases, throughput decreases

**Figure 6.42** Ridges of temporal locality in the memory mountain. The graph shows a slice through Figure 6.41 with stride = 8.
Summary

- A \((S, E, B, m)\) cache \(C\) with memory of size of \(2^m\) has
  - \(S\) cache sets
  - Each set contains \(E\) cache lines
  - Each line contains:
    - 1 data block \(B\) of \(2^b\) bytes, 1 valid bit \(v\), \(t\) tag bits, where \(t = m - (b + s)\)
  - Capacity of \(C = S \times E \times B\)
- Memory address:
- To find data in cache:
  1. **Set Selection**: Select cache set using set index
  2. **Line Matching**: If \(v\) valid, search set for tag
    -> cache hit or cache miss
  3. **Word extraction**: When hit, returns content of block at offset 0 to CPU
No next lecture 😊

- Final Examination
  - Friday April 24 from noon to 3pm
  - Online

- Will soon post Final Examination Instructions on our course web site
- List of Practice Problems