Inside out of your computer memories (II)

Hung-Wei Tseng
Why memory hierarchy?

The access time of DDR3-1600 DRAM is around 50ns 100x to the cycle time of a 2GHz processor!

SRAM is as fast as the processor, but $$$
Memory hierarchy

- **CPU**
  - Fastest, Most Expensive
  - Access time: < 1ns

- **Cache**
  - Access time: < 1ns ~ 20 ns

- **Main Memory**
  - 50-60 ns

- **Secondary Storage**
  - 10,000,000 ns

- **Biggest**
Locality

- Temporal Locality
  - Referenced item tends to be referenced again soon.

- Spatial Locality
  - Items close by referenced item tends to be referenced soon.
  - example: consecutive instructions, arrays
The structure of a cache

**Set:** cache blocks/lines sharing the same index. A cache is called N-way set associative cache if N blocks share the same set/index (this one is a 2-way set cache)

**Block / Cacheline:** The basic unit of data storage in cache. Contains all data with the same tag/prefix and index in their memory addresses

**Tag:** the high order address bits stored along with the data in a block to identify the actual address of the cache line.

**Valid:** if the data is meaningful

**Dirty:** if the block is modified

<table>
<thead>
<tr>
<th>Valid</th>
<th>dirty</th>
<th>tag</th>
<th>data</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>1 0 1000 0001 0000 1000 0000</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Valid</th>
<th>dirty</th>
<th>tag</th>
<th>data</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>1 1 1000 0000 0000 0000 0000</td>
<td></td>
</tr>
</tbody>
</table>
### Accessing the cache

#### Memory Address: 0x8 0 0 0 0 0 1 5 8

**Hit? Miss?**

- **Hit:** The data was found in the cache
- **Miss:** The data was not found in the cache

**Offset:** The position of the requesting word in a cache block

**Tag:** 1000 0000 0000 0000 0000

**Index:** 0001 0101 1000

---

**Valid**

**Dirty**

**Tag**

**Data**

---

**Memory Address:** 0x8 0 0 0 0 0 1 5 8

---

**Hit? Miss?**

---

**Offset:**

---

**Hit: The data was found in the cache**

**Miss: The data was not found in the cache**
C = ABS

- C: Capacity in data arrays
- A: Way-Associativity
  - N-way: N blocks in a set, A = N
  - 1 for direct-mapped cache
- B: Block Size (Cacheline)
  - How many bytes in a block
- S: Number of Sets:
  - A set contains blocks sharing the same index
  - 1 for fully associative cache
Corollary of $C = \text{ABS}$

- offset bits: $\lg(B)$
- index bits: $\lg(S)$
- tag bits: $\text{address\_length} - \lg(S) - \lg(B)$
  - $\text{address\_length}$ is 32 bits for 32-bit machine
- $(\text{address} / \text{block\_size}) \mod S = \text{set index}$
AMD Phenom II

• L1 data (D-L1) cache configuration of AMD Phenom II
  • Size 64KB, 2-way set associativity, 64B block
  • Assume 64-bit memory address

Which of the following is correct?

A. Tag is 49 bits
B. Index is 8 bits
C. Offset is 7 bits
D. The cache has 1024 sets
E. None of the above

C = ABS
64KB = 2 * 64 * S
S = 512
offset = lg(64) = 6 bits
index = lg(512) = 9 bits
tag = 64 - lg(512) - lg(64) = 49 bits
Outline

• How cache/memory hierarchy interacts with CPU
• Performance evaluation considering cache
• Cause of misses
How cache interacts with CPU
What happens on a read?

- Read hit
- hit time

CPU

L1 $

lw

L2 $

tag index offset
What happens on a read?

- Read hit
- hit time
- Read miss?
- Select victim block
  - LRU, random, FIFO, ...
  - Write back if dirty
What happens on a read?

- Read hit
- hit time
- Read miss?
  - Select victim block
    - LRU, random, FIFO, ...
    - Write back if dirty
  - Fetch Data from Lower Memory Hierarchy
    - As a unit of a cache block
      - Data with the same “block address” will be fetch
    - Miss penalty
What happens on a write? (Write Allocate, write back)

- Write hit?
- Update in-place
- Set dirty bit (Write-Back Policy)
What happens on a write? (Write Allocate, write back)

- **Write hit?**
  - Update in-place
  - Set dirty bit (Write-Back Policy)
- **Write miss?**
  - Select victim block
    - LRU, random, FIFO, ...
  - Write back to lower memory hierarchy if dirty
- Fetch Data from Lower Memory Hierarchy
  - As a unit of a cache block
  - Miss penalty
Simulate a 2-way cache

- Consider a 2-way cache with 16 blocks (8 sets), a block size of 16 bytes, and the application repeatedly reading the following memory addresses:
  - 0b1000000000, 0b1000001000, 0b1000010000, 0b1000010100, 0b1100010000
  - 8 = 2^3 : 3 bits are used for the index
  - 16 = 2^4 : 4 bits are used for the byte offset
  - The tag is 32 - (3 + 4) = 25 bits
  - For example: 0b1000 0000 0000 0000 0000 0000 0000 0001 0000

\[ \text{tag} \quad \text{index} \quad \text{offset} \]
Simulate a 2-way cache

<table>
<thead>
<tr>
<th>v</th>
<th>tag</th>
<th>data</th>
<th>v</th>
<th>tag</th>
<th>data</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>0b100</td>
<td>1</td>
<td>0b110</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0b100</td>
<td>1</td>
<td>0b110</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td></td>
<td></td>
<td>2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td></td>
<td></td>
<td>3</td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td></td>
<td></td>
<td>4</td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td></td>
<td></td>
<td>5</td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td></td>
<td></td>
<td>6</td>
<td></td>
<td></td>
</tr>
<tr>
<td>7</td>
<td></td>
<td></td>
<td>7</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>tag</th>
<th>index</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b10 0000 0000</td>
<td>miss</td>
</tr>
<tr>
<td>0b10 0000 1000</td>
<td>hit!</td>
</tr>
<tr>
<td>0b10 0001 0000</td>
<td>miss</td>
</tr>
<tr>
<td>0b10 0001 0100</td>
<td>hit!</td>
</tr>
<tr>
<td>0b10 0001 0100</td>
<td>hit!</td>
</tr>
<tr>
<td>0b11 0001 0000</td>
<td>miss</td>
</tr>
<tr>
<td>0b10 0000 0000</td>
<td>hit!</td>
</tr>
<tr>
<td>0b10 0000 1000</td>
<td>hit!</td>
</tr>
<tr>
<td>0b10 0001 0000</td>
<td>hit!</td>
</tr>
<tr>
<td>0b10 0001 0100</td>
<td>hit!</td>
</tr>
</tbody>
</table>
Special case: a direct-mapped cache

| memory address: | 1000 0000 0000 0000 0000 | 0001 01 01 1000 |

**Tag:**
the high order address bits stored along with the data to identify the actual address of the cache line.

**Block (cacheline):** The basic unit of data storage in cache. Contains all data with the same tag and index in their address.

**Hit:** The data was found in the cache

**Miss:** The data was not found in the cache
Simulate a direct-mapped cache

• Consider a direct mapped (1-way) cache with 16 blocks, a block size of 16 bytes, and the application repeatedly reading the following memory addresses:
  • 0b1000000000, 0b1000001000, 0b1000010000, 0b1000010100, 0b1100010000

• $16 = 2^4$: 4 bits are used for the index
• $16 = 2^4$: 4 bits are used for the byte offset
• The tag is $32 - (4 + 4) = 24$ bits
• For example: 0b1000 0000 0000 0000 0000 0000 1000 0000

  tag

  index

  offset
Simulate a direct-mapped cache

<table>
<thead>
<tr>
<th>valid</th>
<th>tag</th>
<th>data</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0b10</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>0b10</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>tag</th>
<th>index</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b10</td>
<td>0000 0000  miss</td>
</tr>
<tr>
<td>0b10</td>
<td>0000 1000  hit!</td>
</tr>
<tr>
<td>0b10</td>
<td>0001 0000  miss</td>
</tr>
<tr>
<td>0b10</td>
<td>0001 0100  hit!</td>
</tr>
<tr>
<td>0b11</td>
<td>0001 0000  miss</td>
</tr>
<tr>
<td>0b10</td>
<td>0000 0000  hit!</td>
</tr>
<tr>
<td>0b10</td>
<td>0000 1000  hit!</td>
</tr>
<tr>
<td>0b10</td>
<td>0001 0000  miss</td>
</tr>
<tr>
<td>0b10</td>
<td>0001 0100  hit!</td>
</tr>
</tbody>
</table>
Conflict in direct-mapped cache

If we have two frequently used cache blocks:

If they are usually used back-to-back, one will kick out the other all the time
• **D-L1 Cache configuration of AMD Phenom II**
  • Size 64KB, 2-way set associativity, 64B block, LRU policy, write-allocate, write-back, and assuming 32-bit address.
  • Cache performance for the following code?
    • int a[16384], b[16384], c[16384];
      /* c = 0x10000, a = 0x20000, b = 0x30000 */
      for(i = 0; i < 512; i++) {
        c[i] = a[i] + b[i];
        //load a, b, and then store to c
      }
    • What’s the data cache miss rate for this code?
      A. 6.25%  
      B. 56.25%  
      C. 66.67%  
      D. 68.75%  
      E. 100%  

- $C = \text{ABS}$
- $64\text{KB} = 2 * 64 * S$
- $S = 512$
- $\text{offset} = \lg(64) = 6 \text{ bits}$
- $\text{index} = \lg(512) = 9 \text{ bits}$
- $\text{tag} = 64 - \lg(512) - \lg(64) = 49 \text{ bits}$
```c
int a[16384], b[16384], c[16384];
/* c = 0x10000, a = 0x20000, b = 0x30000 */
for(i = 0; i < 512; i++)
{
    c[i] = a[i] + b[i]; /*load a[i], load b[i], store c[i]*/
}
```

<table>
<thead>
<tr>
<th>address</th>
<th>tag</th>
<th>index</th>
<th>hit? miss?</th>
</tr>
</thead>
<tbody>
<tr>
<td>load a[0]</td>
<td>0x20000</td>
<td>0x4</td>
<td>0</td>
</tr>
<tr>
<td>load b[0]</td>
<td>0x30000</td>
<td>0x6</td>
<td>0</td>
</tr>
<tr>
<td>load c[0]</td>
<td>0x10000</td>
<td>0x2</td>
<td>0</td>
</tr>
<tr>
<td>load a[1]</td>
<td>0x20004</td>
<td>0x4</td>
<td>0</td>
</tr>
<tr>
<td>load b[1]</td>
<td>0x30004</td>
<td>0x6</td>
<td>0</td>
</tr>
<tr>
<td>load c[1]</td>
<td>0x10004</td>
<td>0x2</td>
<td>0</td>
</tr>
</tbody>
</table>

100% miss rate!
intel Core i7

- D-L1 Cache configuration of intel Core i7 processor
  - Size 32KB, 8-way set associativity, 64B block, LRU policy, write-allocate, write-back, and assuming 64-bit address.

Cache performance for the following code?

```cpp
int a[16384], b[16384], c[16384];
/* c = 0x10000, a = 0x20000, b = 0x30000 */
for(i = 0; i < 512; i++) {
    c[i] = a[i] + b[i];
    //load a, b, and then store to c
}
```

- What’s the data cache miss rate for this code?
  - A. 6.25%
  - B. 56.25%
  - C. 66.67%
  - D. 68.75%
  - E. 100%

C = ABS

32KB = 8 * 64 * S
S = 64

offset = lg(64) = 6 bits
index = lg(64) = 6 bits
tag = 64 - lg(64) - lg(64) = 52 bits
intel Core i7

```c
int a[16384], b[16384], c[16384];
/* c = 0x10000, a = 0x20000, b = 0x30000 */
for(i = 0; i < 512; i++)
{
    c[i] = a[i] + b[i]; /*load a[i], load b[i], store c[i]*/
}
```

<table>
<thead>
<tr>
<th>address</th>
<th>tag</th>
<th>index</th>
<th>?</th>
</tr>
</thead>
<tbody>
<tr>
<td>load a[0]</td>
<td>0x20000</td>
<td>0x20</td>
<td>0</td>
</tr>
<tr>
<td>load b[0]</td>
<td>0x30000</td>
<td>0x30</td>
<td>0</td>
</tr>
<tr>
<td>load c[0]</td>
<td>0x10000</td>
<td>0x10</td>
<td>0</td>
</tr>
<tr>
<td>load a[1]</td>
<td>0x20004</td>
<td>0x20</td>
<td>0</td>
</tr>
<tr>
<td>load b[1]</td>
<td>0x30004</td>
<td>0x30</td>
<td>0</td>
</tr>
<tr>
<td>load c[1]</td>
<td>0x10004</td>
<td>0x10</td>
<td>0</td>
</tr>
</tbody>
</table>

6.25% miss rate!
Performance evaluation considering cache
Performance evaluation considering cache

- If the load/store instruction hits in L1 cache where the hit time is usually the same as a CPU cycle
  - The CPI of this instruction is the base CPI
- If the load/store instruction misses in L1, we need to access L2
  - The CPI of this instruction needs to include the cycles of accessing L2
- If the load/store instruction misses in both L1 and L2, we need to go to lower memory hierarchy (L3 or DRAM)
  - The CPI of this instruction needs to include the cycles of accessing L2, L3, DRAM
How to evaluate cache performance

- CPI\text{Average} : the average CPI of a memory instruction

\[ \text{CPI} \text{Average} = \text{CPI}_\text{base} + \text{miss\_rate}_{L1} \times \text{miss\_penalty}_{L1} \]

\[ \text{miss\_penalty}_{L1} = \text{CPI}_\text{accessing\_L2} + \text{miss\_rate}_{L2} \times \text{miss\_penalty}_{L2} \]

\[ \text{miss\_penalty}_{L2} = \text{CPI}_\text{accessing\_L3} + \text{miss\_rate}_{L3} \times \text{miss\_penalty}_{L3} \]

\[ \text{miss\_penalty}_{L3} = \text{CPI}_\text{accessing\_DRAM} + \text{miss\_rate}_{DRAM} \times \text{miss\_penalty}_{DRAM} \]

- If the problem is asking for average memory access time, transform the CPI values into/from time by multiplying with CPU cycle time!
Average memory access time

• Average Memory Access Time (AMAT) = Hit Time + Miss rate* Miss penalty
• Miss penalty = AMAT of the lower memory hierarchy
• AMAT = hit_time_{L1} + miss_rate_{L1} * AMAT_{L2}
  • AMAT_{L2} = hit_time_{L2} + miss_rate_{L2} * AMAT_{DRAM}
Cache & Performance

• 5-stage MIPS processor.
  • Application: 80% ALU, 20% Loads
  • L1 I-cache miss rate: 5%, hit time: 1 cycle
  • L1 D-cache miss rate: 10%, hit time: 1 cycle
  • L2 U-Cache miss rate: 20%, hit time: 10 cycles
  • Main memory hit time: 100 cycles
  • Assume the program is read only (nothing dirty)
  • What’s the average CPI?

A. 1.1
B. 1.6
C. 2.1
D. 3.1
E. none of the above

\[
\text{CPI}_{\text{Average}} = \text{CPI}_{\text{base}} + \text{miss\_rate} \times \text{miss\_penalty}
\]

\[
= 1 + 100\% \times (5\% \times (10 + 20\% \times (1 \times 100)))
\]

\[
+ 20\% \times (10\% \times (10 + 20\% \times ((1) \times 100)))
\]

\[
= 3.1
\]
Cache & Performance

- Application: 80% ALU, 20% Loads
- L1 I-cache miss rate: 5%, hit time: 1 cycle
- L1 D-cache miss rate: 10%, hit time: 1 cycle
- L2 U-Cache miss rate: 20%, hit time: 10 cycles
- Main memory hit time: 100 cycles
- What’s the average CPI?

CPI\text{Average} = CPI_{\text{base}} + \text{miss}_\text{rate} \times \text{miss}_\text{penalty}

= 1 + 100\% \times (5\% \times (10 + 20\% \times (1 \times 100))) + 20\% \times (10\% \times (10 + 20\% \times (1 \times 100)))

= 3.1
Cache & Performance

• 5-stage MIPS processor.
  • Application: 80% ALU, 20% Loads and stores
  • L1 I-cache miss rate: 5%, hit time: 1 cycle
  • L1 D-cache miss rate: 10%, hit time: 1 cycle, 20% of the replaced blocks are dirty.
  • L2 U-Cache miss rate: 20%, hit time: 10 cycles, 10% of the replaced blocks are dirty.
  • Main memory hit time: 100 cycles
  • What’s the average CPI?

A. 0.77
B. 2.6
C. 3.37
D. 4.1
E. none of the above
Cache & Performance

- Application: 80% ALU, 20% Load/Store
- L1 I-cache miss rate: 5%, hit time: 1 cycle
- L1 D-cache miss rate: 10%, hit time: 1 cycle, 20% dirty
- L2 U-Cache miss rate: 20%, hit time: 10 cycles, 10% dirty
- Main memory hit time: 100 cycles
- What's the average CPI?

\[
\text{CPI}_{\text{Average}} = \text{CPI}_{\text{base}} + \text{miss}\_\text{rate} \times \text{miss}\_\text{penalty}
\]

\[
= 1 + 100\% \times (5\% \times (10 + 20\% \times ((1 + 10\%) \times 100))) + 20\% \times (10\% \times (1 + 20\%) \times (10 + 20\% \times ((1 + 10\%) \times 100)))
\]

\[
= 3.368
\]
Announcement

- Homework #4 due next Tuesday
- Don’t forget to register your iclicker