Cache Organization

Mengjia Yan
Computer Science and Artificial Intelligence Laboratory
M.I.T.

Based on slides from Daniel Sanchez
CPU-Memory Bottleneck
CPU-Memory Bottleneck

Performance of high-speed computers is usually
CPU-Memory Bottleneck

Performance of high-speed computers is usually limited by memory bandwidth & latency

- Latency (time for a single access)
  Memory access time >> Processor cycle time

- Bandwidth (number of accesses per unit time)
Performance of high-speed computers is usually limited by memory bandwidth & latency

- **Latency** (time for a single access)
  Memory access time $\gg$ Processor cycle time

- **Bandwidth** (number of accesses per unit time)
  if fraction $m$ of instructions access memory,  
  $\Rightarrow 1+m$ memory references / instruction  
  $\Rightarrow$ CPI = 1 requires $1+m$ memory refs / cycle
Memory Technology

• Early machines used a variety of memory technologies
  – Manchester Mark I used CRT Memory Storage
  – EDVAC used a mercury delay line

• Core memory was first large scale reliable main memory
  – Invented by Forrester in late 40s at MIT for Whirlwind project
  – Bits stored as magnetization polarity on small ferrite cores threaded onto 2 dimensional grid of wires
Memory Technology

• Early machines used a variety of memory technologies
  – Manchester Mark I used CRT Memory Storage
  – EDVAC used a mercury delay line

• Core memory was first large scale reliable main memory
  – Invented by Forrester in late 40s at MIT for Whirlwind project
  – Bits stored as magnetization polarity on small ferrite cores threaded onto 2 dimensional grid of wires

• First commercial DRAM was Intel 1103
  – 1Kbit of storage on single chip
  – charge on a capacitor used to hold value

• Semiconductor memory quickly replaced core in 1970s
  – Intel formed to exploit market for semiconductor memory
Memory Technology

- Early machines used a variety of memory technologies
  - Manchester Mark I used CRT Memory Storage
  - EDVAC used a mercury delay line

- Core memory was first large scale reliable main memory
  - Invented by Forrester in late 40s at MIT for Whirlwind project
  - Bits stored as magnetization polarity on small ferrite cores threaded onto 2 dimensional grid of wires

- First commercial DRAM was Intel 1103
  - 1Kbit of storage on single chip
  - charge on a capacitor used to hold value

- Semiconductor memory quickly replaced core in 1970s
  - Intel formed to exploit market for semiconductor memory

- Flash memory
  - Slower, but denser than DRAM. Also non-volatile, but with wearout issues

- Phase change memory (PCM, 3D XPoint)
  - Slightly slower, but much denser than DRAM and non-volatile
DRAM Architecture

Memory cell (one bit)
Bits stored in 2-dimensional arrays on chip
Bits stored in 2-dimensional arrays on chip

Question: why read the entire row?
DRAM Architecture

- Bits stored in 2-dimensional arrays on chip
- Question: why read the entire row?
- Modern chips have around 8 logical banks on each chip
  - each logical bank physically implemented as many smaller arrays
DRAM timing

DRAM Spec: CL, tRCD, tRP, tRAS, e.g., 9-9-9-24
Processor-DRAM Gap (latency)
Four-issue 2GHz superscalar accessing 100ns DRAM could execute 800 instructions during time for one memory access!
Little’s Law

Throughput \((T) = \frac{\text{Number in Flight (N)}}{\text{Latency (L)}}\)

Example:

--- Assume infinite-bandwidth memory
--- 100 cycles / memory reference
--- 1 + 0.2 memory references / instruction
Little’s Law

Throughput \((T) = \frac{\text{Number in Flight (N)}}{\text{Latency (L)}}\)

Example:

--- Assume infinite-bandwidth memory
--- 100 cycles / memory reference
--- 1 + 0.2 memory references / instruction

\[ \Rightarrow \text{Table size} = 1.2 \times 100 = 120 \text{ entries} \]

120 independent memory operations in flight!
Basic Static RAM Cell

Write:
1. Drive bit lines (bit=1, \( \overline{\text{bit}} \)=0)
2. Select word line

Read:
1. Precharge bit and \( \overline{\text{bit}} \) to Vdd
2. Select word line
3. Cell pulls one bit line low
4. Column sense amp detects difference between bit & \( \overline{\text{bit}} \)
Multilevel Memory

Strategy: **Reduce** average latency using small, fast memories called caches.

Caches are a mechanism to reduce memory latency based on the empirical observation that the patterns of memory references made by a processor are often highly predictable:
Multilevel Memory

Strategy: **Reduce** average latency using small, fast memories called caches.

Caches are a mechanism to reduce memory latency based on the empirical observation that the patterns of memory references made by a processor are often highly predictable:

```
...                      PC
                  96
Loop:          add r2, r1, r1   100
              subi r3, r3, #1    104
              bnez r3, loop      108
...                      112
...                      ...
```

February 11, 2020
Typical Memory Reference Patterns

Address

Instruction fetches

n loop iterations

Time

February 11, 2020
Typical Memory Reference Patterns

Address

Instruction fetches

Stack accesses

n loop iterations

Time
Typical Memory Reference Patterns

Address

Instruction fetches

Stack accesses

Time

n loop iterations
Typical Memory Reference Patterns

Address \[\text{n loop iterations}\] Time

Instruction fetches
Stack accesses

subroutine call
Typical Memory Reference Patterns

Address

Instruction fetches

Stack accesses

n loop iterations

subroutine call

argument access

Time
Typical Memory Reference Patterns

- Instruction fetches
- Stack accesses

Address vs. Time

- n loop iterations
- Subroutine call
- Argument access
- Subroutine return

L03-10
Typical Memory Reference Patterns

- Instruction fetches
- Stack accesses
- Data accesses

Address

Time

n loop iterations

subroutine call

argument access

subroutine return
Typical Memory Reference Patterns

- Instruction fetches
- Stack accesses
- Data accesses

Address vs. Time

n loop iterations

subroutine call
argument access
subroutine return
Typical Memory Reference Patterns

- Instruction fetches
- Stack accesses
- Data accesses

Address

Time

n loop iterations

subroutine call

argument access

vector access

subroutine return
Typical Memory Reference Patterns

Address

Instruction fetches

Stack accesses

Data accesses

Time

n loop iterations

subroutine call

argument access

vector access

scalar accesses

subroutine return
Common Predictable Patterns

Two predictable properties of memory references:

– *Temporal Locality*: If a location is referenced it is likely to be referenced again in the near future.

– *Spatial Locality*: If a location is referenced it is likely that locations near it will be referenced in the near future.
Memory Hierarchy

- **size**: Register $<<$ SRAM $<<$ DRAM  why?
- **latency**: Register $<<$ SRAM $<<$ DRAM  why?
- **bandwidth**: on-chip $>>$ off-chip  why?

Small, Fast Memory (RF, SRAM)

holds frequently used data

CPU

Big, Slow Memory (DRAM)
Memory Hierarchy

- **size:** Register $<<$ SRAM $<<$ DRAM  why?
- **latency:** Register $<<$ SRAM $<<$ DRAM  why?
- **bandwidth:** on-chip $>>$ off-chip  why?

On a data access:
- *hit* (data $\in$ fast memory) $\Rightarrow$ low latency access
- *miss* (data $\notin$ fast memory) $\Rightarrow$ long latency access *(DRAM)*
Typical memory hierarchies

(a) Memory hierarchy for server

- CPU: Registers
  - Register reference
- L1 Cache: Level 1 Cache reference
  - Size: 1000 bytes
  - Speed: 300 ps
- L2 Cache: Level 2 Cache reference
  - Size: 64 KB
  - Speed: 1 ns
- L3 Cache: Level 3 Cache reference
  - Size: 256 KB
  - Speed: 3–10 ns
- Memory: Memory reference
  - Size: 2–4 MB
  - Speed: 10–20 ns
- Memory bus
- I/O bus
- Disk storage
  - Size: 4–16 GB
  - Speed: 50–100 ns
  - Memory reference
  - Size: 4–16 TB
  - Speed: 5–10 ms

(b) Memory hierarchy for a personal mobile device

- CPU: Registers
  - Register reference
- L1 Cache: Level 1 Cache reference
  - Size: 500 bytes
  - Speed: 500 ps
- L2 Cache: Level 2 Cache reference
  - Size: 64 KB
  - Speed: 2 ns
- Memory: Memory reference
  - Size: 256 KB
  - Speed: 10–20 ns
- Memory bus
- Storage
  - Flash memory reference
  - Size: 256–512 MB
  - Speed: 50–100 ns
  - Memory reference
  - Size: 4–8 GB
  - Speed: 25–50 us
Management of Memory Hierarchy

- **Small/fast storage, e.g., registers**
  - Address usually specified in instruction
  - Generally implemented directly as a register file
    - but hardware might do things behind software’s back, e.g., stack management, register renaming

- **Large/slower storage, e.g., memory**
  - Address usually computed from values in register
  - Generally implemented as a cache hierarchy
    - hardware decides what is kept in fast memory
    - but software may provide “hints”, e.g., don’t cache or prefetch
Inside a Cache

Processor ➔ Address ➔ CACHE ➔ Address ➔ Main Memory
Data ➔ CACHE ➔ Data

- Copy of main memory location 100
- Copy of main memory location 101

<table>
<thead>
<tr>
<th>Address</th>
<th>Data Byte</th>
<th>Data Byte</th>
<th>Data Block</th>
</tr>
</thead>
<tbody>
<tr>
<td>100</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>304</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>6848</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>416</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Address Tag

Line

Data Block
Inside a Cache

Q: How many bits needed in tag? ___________________________
Q: Why not use very small/big block size?
Inside a Cache

Q: How many bits needed in tag? **Enough to uniquely identify block**
Q: Why not use very small/big block size?
Direct-Mapped Cache

\[
\begin{array}{c|c|c}
V & \text{Tag} & \text{Data Block} \\
\hline
\vdots & \vdots & \vdots \\
\vdots & \vdots & \vdots \\
\vdots & \vdots & \vdots \\
\vdots & \vdots & \vdots \\
\end{array}
\]

\[2^k\text{ lines}\]
Direct-Mapped Cache

![Diagram of Direct-Mapped Cache]

- **Tag**: Stores the tag of the address
- **Index**: Computes the index from the tag
- **Block number** and **Block offset**: Together form the address
- **Data Block**: Stores the data

2^k lines

V | Tag | Data Block
---|-----|-----------------
Direct-Mapped Cache

Block number

Tag

Index

Offset

Data Block

\(2^k\) lines

Tag number

\(k\)
Direct-Mapped Cache

Block number

Block offset

Tag | Index | Offset

2^k lines

HIT
Direct-Mapped Cache

Block number

Block offset

Tag

Index

Offset

Data Word or Byte

HIT

$V_k$
Direct-Mapped Cache

Q: What is a bad reference pattern? ____________________
Q: What is a bad reference pattern?  

Strided at size of cache
Direct Map Address Selection
higher-order vs. lower-order address bits

Q: Why might this be undesirable? ________________________
Direct Map Address Selection
higher-order vs. lower-order address bits

Q: Why might this be undesirable? Spatially local blocks conflict
Hashed Address Mapping

Q: What are the tradeoffs of hashing?
**Hashed Address Mapping**

**Q: What are the tradeoffs of hashing?**

**Good:** Regular strides don’t conflict
**Bad:** Hash adds latency
Tag is larger
2-Way Set-Associative Cache

<table>
<thead>
<tr>
<th>Tag</th>
<th>Index</th>
<th>Block Offset</th>
</tr>
</thead>
</table>

V Tag | Data Block
--- | ---
. | . | .

V Tag | Data Block
--- | ---
. | . | .

. | .
2-Way Set-Associative Cache

- Tag
- Index
- Block Offset

V Tag Data Block

V Tag Data Block

k
2-Way Set-Associative Cache

- Tag
- Index
- Block Offset

V Tag Data Block

HIT
2-Way Set-Associative Cache

Diagram illustrating a 2-way set-associative cache with tag, index, and block offset fields. The diagram shows how data blocks are accessed and matched for hits.
Fully Associative Cache
Placement Policy

Block Number

Memory

Set Number

Cache

Direct Mapped only into block 4 \((12 \ mod \ 8)\)

block 12 can be placed
Placement Policy

Block Number
0 1 2 3 4 5 6 7 8 9

Memory

Set Number
0 1 2 3 4 5 6 7

Cache

Direct Mapped only into block 4 \((12 \mod 8)\)

(2-way) SetAssociative anywhere in set 0 \((12 \mod 4)\)

block 12 can be placed only into block 4 \((12 \mod 8)\)

\((12 \mod 4)\)
Placement Policy

Memory

Block Number
0 1 2 3 4 5 6 7 8 9

Set Number
0 1 2 3 4 5 6 7

Cache

0 1 2 3 4 5 6 7

Direct Mapped only into block 4
(12 mod 8)

(2-way) Set Associative anywhere in set 0
(12 mod 4)

Fully Associative anywhere

block 12 can be placed

block 12
(12 mod 8)

MIT 6.823 Spring 2020

February 11, 2020

L03-21
Cache Algorithm (Read)

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

Which line do we replace?
Improving Cache Performance

Average memory access time = Hit time + Miss rate x Miss penalty

To improve performance:
• reduce the hit time
• reduce the miss rate (e.g., larger, better policy)
• reduce the miss penalty (e.g., L2 cache)

What is the simplest design strategy?
Improving Cache Performance

Average memory access time = 
Hit time + Miss rate x Miss penalty

To improve performance:
• reduce the hit time
• reduce the miss rate (e.g., larger, better policy)
• reduce the miss penalty (e.g., L2 cache)

What is the simplest design strategy?

Biggest cache that doesn’t increase hit time past 1-2 cycles 
(approx. 16-64KB in modern technology)
[design issues more complex with out-of-order superscalar processors]
Causes for Cache Misses

- **Compulsory:**
  First reference to a block *a.k.a.* cold start misses
  - misses that would occur even with infinite cache

- **Capacity:**
  cache is too small to hold all data the program needs
  - misses that would occur even under perfect placement & replacement policy

- **Conflict:**
  misses from collisions due to block-placement strategy
  - misses that would not occur with full associativity
Effect of Cache Parameters on Performance

<table>
<thead>
<tr>
<th></th>
<th>Larger capacity cache</th>
<th>Higher associativity cache</th>
<th>Larger block size cache *</th>
</tr>
</thead>
<tbody>
<tr>
<td>Compulsory misses</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Capacity misses</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Conflict misses</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Hit latency</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Miss latency</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

* Assume substantial spatial locality
## Effect of Cache Parameters on Performance

<table>
<thead>
<tr>
<th></th>
<th>Larger capacity cache</th>
<th>Higher associativity cache</th>
<th>Larger block size cache *</th>
</tr>
</thead>
<tbody>
<tr>
<td>Compulsory misses</td>
<td>🔄</td>
<td>🔄</td>
<td>🔄</td>
</tr>
<tr>
<td>Capacity misses</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Conflict misses</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Hit latency</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Miss latency</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

* Assume substantial spatial locality
# Effect of Cache Parameters on Performance

<table>
<thead>
<tr>
<th></th>
<th>Larger capacity cache</th>
<th>Higher associativity cache</th>
<th>Larger block size cache *</th>
</tr>
</thead>
<tbody>
<tr>
<td>Compulsory misses</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Capacity misses</td>
<td></td>
<td>![Red Down Arrow]</td>
<td></td>
</tr>
<tr>
<td>Conflict misses</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Hit latency</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Miss latency</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

* Assume substantial spatial locality
## Effect of Cache Parameters on Performance

<table>
<thead>
<tr>
<th></th>
<th>Larger capacity cache</th>
<th>Higher associativity cache</th>
<th>Larger block size cache *</th>
</tr>
</thead>
<tbody>
<tr>
<td>Compulsory misses</td>
<td>=</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Capacity misses</td>
<td>↓</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Conflict misses</td>
<td>↓</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Hit latency</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Miss latency</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

* Assume substantial spatial locality
## Effect of Cache Parameters on Performance

<table>
<thead>
<tr>
<th></th>
<th>Larger capacity cache</th>
<th>Higher associativity cache</th>
<th>Larger block size cache *</th>
</tr>
</thead>
<tbody>
<tr>
<td>Compulsory misses</td>
<td>≈</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Capacity misses</td>
<td>↓</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Conflict misses</td>
<td>↓</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Hit latency</td>
<td>↑</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Miss latency</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

* Assume substantial spatial locality
# Effect of Cache Parameters on Performance

<table>
<thead>
<tr>
<th></th>
<th>Larger capacity cache</th>
<th>Higher associativity cache</th>
<th>Larger block size cache *</th>
</tr>
</thead>
<tbody>
<tr>
<td>Compulsory misses</td>
<td>=</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Capacity misses</td>
<td>↓</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Conflict misses</td>
<td>↓</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Hit latency</td>
<td>↑</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Miss latency</td>
<td>=</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

* Assume substantial spatial locality
## Effect of Cache Parameters on Performance

<table>
<thead>
<tr>
<th></th>
<th>Larger capacity cache</th>
<th>Higher associativity cache</th>
<th>Larger block size cache *</th>
</tr>
</thead>
<tbody>
<tr>
<td>Compulsory misses</td>
<td>=</td>
<td>=</td>
<td></td>
</tr>
<tr>
<td>Capacity misses</td>
<td>↓</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Conflict misses</td>
<td>↓</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Hit latency</td>
<td>↑</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Miss latency</td>
<td>=</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

* Assume substantial spatial locality
# Effect of Cache Parameters on Performance

<table>
<thead>
<tr>
<th></th>
<th>Larger capacity cache</th>
<th>Higher associativity cache</th>
<th>Larger block size cache *</th>
</tr>
</thead>
<tbody>
<tr>
<td>Compulsory misses</td>
<td>=</td>
<td>=</td>
<td>=</td>
</tr>
<tr>
<td>Capacity misses</td>
<td>↓</td>
<td>=</td>
<td>=</td>
</tr>
<tr>
<td>Conflict misses</td>
<td>↓</td>
<td>=</td>
<td>=</td>
</tr>
<tr>
<td>Hit latency</td>
<td>↑</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Miss latency</td>
<td>=</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

* Assume substantial spatial locality
### Effect of Cache Parameters on Performance

<table>
<thead>
<tr>
<th></th>
<th>Larger capacity cache</th>
<th>Higher associativity cache</th>
<th>Larger block size cache *</th>
</tr>
</thead>
<tbody>
<tr>
<td>Compulsory misses</td>
<td>=</td>
<td>=</td>
<td>=</td>
</tr>
<tr>
<td>Capacity misses</td>
<td>↓</td>
<td>=</td>
<td>=</td>
</tr>
<tr>
<td>Conflict misses</td>
<td>↓</td>
<td>↓</td>
<td>↓</td>
</tr>
<tr>
<td>Hit latency</td>
<td>↑</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Miss latency</td>
<td>=</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

* Assume substantial spatial locality
# Effect of Cache Parameters on Performance

<table>
<thead>
<tr>
<th></th>
<th>Larger capacity cache</th>
<th>Higher associativity cache</th>
<th>Larger block size cache *</th>
</tr>
</thead>
<tbody>
<tr>
<td>Compulsory misses</td>
<td>=</td>
<td>=</td>
<td>=</td>
</tr>
<tr>
<td>Capacity misses</td>
<td>▼</td>
<td>=</td>
<td>=</td>
</tr>
<tr>
<td>Conflict misses</td>
<td>▼</td>
<td>▼</td>
<td>▼</td>
</tr>
<tr>
<td>Hit latency</td>
<td>▲</td>
<td>▲</td>
<td>▲</td>
</tr>
<tr>
<td>Miss latency</td>
<td>=</td>
<td>=</td>
<td>=</td>
</tr>
</tbody>
</table>

* Assume substantial spatial locality
Effect of Cache Parameters on Performance

|                              | Larger capacity cache | Higher associativity cache | Larger block size cache *
|------------------------------|------------------------|-----------------------------|-----------------------------
| Compulsory misses           | =                      | =                          | =                          |
| Capacity misses             | ↓                      | =                          | =                          |
| Conflict misses             | ↓                      | ↓                          | ↓                          |
| Hit latency                 | ↑                      | ↑                          | ↑                          |
| Miss latency                | =                      | =                          | =                          |

* Assume substantial spatial locality
# Effect of Cache Parameters on Performance

<table>
<thead>
<tr>
<th></th>
<th>Larger capacity cache</th>
<th>Higher associativity cache</th>
<th>Larger block size cache *</th>
</tr>
</thead>
<tbody>
<tr>
<td>Compulsory misses</td>
<td>=</td>
<td>=</td>
<td>↓</td>
</tr>
<tr>
<td>Capacity misses</td>
<td>↓</td>
<td>=</td>
<td></td>
</tr>
<tr>
<td>Conflict misses</td>
<td>↓</td>
<td>↓</td>
<td></td>
</tr>
<tr>
<td>Hit latency</td>
<td>↑</td>
<td>↑</td>
<td></td>
</tr>
<tr>
<td>Miss latency</td>
<td>=</td>
<td>=</td>
<td></td>
</tr>
</tbody>
</table>

* Assume substantial spatial locality
Effect of Cache Parameters on Performance

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Larger capacity cache</th>
<th>Higher associativity cache</th>
<th>Larger block size cache *</th>
</tr>
</thead>
<tbody>
<tr>
<td>Compulsory misses</td>
<td>=</td>
<td>=</td>
<td>□</td>
</tr>
<tr>
<td>Capacity misses</td>
<td>□</td>
<td>=</td>
<td>□</td>
</tr>
<tr>
<td>Conflict misses</td>
<td>□</td>
<td>□</td>
<td>□</td>
</tr>
<tr>
<td>Hit latency</td>
<td>□</td>
<td>□</td>
<td>□</td>
</tr>
<tr>
<td>Miss latency</td>
<td>=</td>
<td>=</td>
<td>□</td>
</tr>
</tbody>
</table>

* Assume substantial spatial locality
### Effect of Cache Parameters on Performance

<table>
<thead>
<tr>
<th></th>
<th>Larger capacity cache</th>
<th>Higher associativity cache</th>
<th>Larger block size cache *</th>
</tr>
</thead>
<tbody>
<tr>
<td>Compulsory misses</td>
<td>(=)</td>
<td>(=)</td>
<td>(\downarrow)</td>
</tr>
<tr>
<td>Capacity misses</td>
<td>(\downarrow)</td>
<td>(=)</td>
<td>(\downarrow)</td>
</tr>
<tr>
<td>Conflict misses</td>
<td>(\downarrow)</td>
<td>(\downarrow)</td>
<td>?</td>
</tr>
<tr>
<td>Hit latency</td>
<td>(\uparrow)</td>
<td>(\uparrow)</td>
<td></td>
</tr>
<tr>
<td>Miss latency</td>
<td>(=)</td>
<td>(=)</td>
<td></td>
</tr>
</tbody>
</table>

* Assume substantial spatial locality
### Effect of Cache Parameters on Performance

<table>
<thead>
<tr>
<th></th>
<th>Larger capacity cache</th>
<th>Higher associativity cache</th>
<th>Larger block size cache *</th>
</tr>
</thead>
<tbody>
<tr>
<td>Compulsory misses</td>
<td>=</td>
<td>=</td>
<td>↓</td>
</tr>
<tr>
<td>Capacity misses</td>
<td>↓</td>
<td>=</td>
<td>↓</td>
</tr>
<tr>
<td>Conflict misses</td>
<td>↓</td>
<td>↓</td>
<td>?</td>
</tr>
<tr>
<td>Hit latency</td>
<td>↑</td>
<td>↑</td>
<td>=</td>
</tr>
<tr>
<td>Miss latency</td>
<td>=</td>
<td>=</td>
<td></td>
</tr>
</tbody>
</table>

* Assume substantial spatial locality
# Effect of Cache Parameters on Performance

<table>
<thead>
<tr>
<th></th>
<th>Larger capacity cache</th>
<th>Higher associativity cache</th>
<th>Larger block size cache *</th>
</tr>
</thead>
<tbody>
<tr>
<td>Compulsory misses</td>
<td>≈</td>
<td>≈</td>
<td>↓</td>
</tr>
<tr>
<td>Capacity misses</td>
<td>↓</td>
<td>≈</td>
<td>↓</td>
</tr>
<tr>
<td>Conflict misses</td>
<td>↓</td>
<td>↓</td>
<td>?</td>
</tr>
<tr>
<td>Hit latency</td>
<td>↑</td>
<td>↑</td>
<td>≈</td>
</tr>
<tr>
<td>Miss latency</td>
<td>≈</td>
<td>≈</td>
<td>↑↓</td>
</tr>
</tbody>
</table>

* Assume substantial spatial locality
Block-level Optimizations

• Tags are too large, i.e., too much overhead
  – Simple solution: Larger blocks, but miss penalty could be large.
### Block-level Optimizations

- **Tags are too large, i.e., too much overhead**
  - Simple solution: Larger blocks, but miss penalty could be large.

- **Sub-block placement (aka sector cache)**
  - A valid bit added to units smaller than the full block, called sub-blocks
  - Only read a sub-block on a miss
  - *If a tag matches, is the sub-block in the cache?*

<table>
<thead>
<tr>
<th></th>
<th>100</th>
<th>300</th>
<th>204</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>
Replacement Policy

Which block from a set should be evicted?

- Random
Replacement Policy

Which block from a set should be evicted?

- Random
Replacement Policy

Which block from a set should be evicted?

- Random

- Least Recently Used (LRU)
  - LRU cache state must be updated on every access
  - true implementation only feasible for small sets (2-way)
  - pseudo-LRU binary tree was often used for 4-8 way
Replacement Policy

Which block from a set should be evicted?

- Random

- Least Recently Used (LRU)
  - LRU cache state must be updated on every access
  - true implementation only feasible for small sets (2-way)
  - pseudo-LRU binary tree was often used for 4-8 way

- First In, First Out (FIFO) a.k.a. Round-Robin
  - used in highly associative caches
Replacement Policy

Which block from a set should be evicted?

- Random

- Least Recently Used (LRU)
  - LRU cache state must be updated on every access
  - true implementation only feasible for small sets (2-way)
  - pseudo-LRU binary tree was often used for 4-8 way

- First In, First Out (FIFO) a.k.a. Round-Robin
  - used in highly associative caches

- Not Least Recently Used (NLRU)
  - FIFO with exception for most recently used block or blocks
Replacement Policy

Which block from a set should be evicted?

- Random

- Least Recently Used (LRU)
  - LRU cache state must be updated on every access
  - true implementation only feasible for small sets (2-way)
  - pseudo-LRU binary tree was often used for 4-8 way

- First In, First Out (FIFO) a.k.a. Round-Robin
  - used in highly associative caches

- Not Least Recently Used (NLRU)
  - FIFO with exception for most recently used block or blocks

- One-bit LRU
  - Each way represented by a bit. Set on use, replace first unused.
Multilevel Caches

- A memory cannot be large and fast
- Add level of cache to reduce miss penalty
  - Each level can have longer latency than level above
  - So, increase sizes of cache at each level

```
CPU <-> L1 <-> L2 <-> DRAM
```

Metrics:

Local miss rate = misses in cache / accesses to cache
Global miss rate = misses in cache / CPU memory accesses
Misses per instruction (MPI) = misses in cache / number of instructions
Victim Caches (HP 7200)

Victim cache is a small associative back up cache, added to a direct mapped cache, which holds recently evicted lines

- First look up in direct mapped cache
- If miss, look in victim cache
- If hit in victim cache, swap hit line with line now evicted from L1
- If miss in victim cache, L1 victim -> VC, VC victim->?

Fast hit time of direct mapped but with reduced conflict misses
Inclusion Policy

• Inclusive multilevel cache:
  – Inner cache holds copies of data in outer cache
  – External access need only check outer cache
  – Most common case

• Exclusive multilevel caches:
  – Inner cache may hold data not in outer cache
  – Swap lines between inner/outer caches on miss
  – Used in AMD Athlon with 64KB primary and 256KB secondary cache

• Non-inclusive multilevel caches:
  – Intel Skylake
  – ARM
Inclusion Policy

• Inclusive multilevel cache:
  – Inner cache holds copies of data in outer cache
  – External access need only check outer cache
  – Most common case

• *Exclusive* multilevel caches:
  – Inner cache may hold data not in outer cache
  – Swap lines between inner/outer caches on miss
  – Used in AMD Athlon with 64KB primary and 256KB secondary cache

• Non-inclusive multilevel caches:
  – Intel Skylake
  – ARM

Why choose one type or the other?
HBM DRAM or MCDRAM

Source: AMD
Mixed technology caching (Intel Knights Landing)
Thank you!

Next lecture: Virtual memory