Caches (continued)

Mengjia Yan
Computer Science and Artificial Intelligence Laboratory
M.I.T.
Reminder: Inside a Cache

- Processor
- CACHE
- Main Memory

Address
Data

copy of main memory location 100

Address
Tag

100
304
6848
416

Data Byte

Data Block

Line
Reminder: 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?
Reminder: Cache Performance

Average memory access time (AMAT) = 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 [Hill, 1989]

• **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 fully-associative placement & perfect 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
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.
Multiple replacement policies

Use the best replacement policy for a program

How do we decide which policy to use?

0: Policy A Missed
1: Policy B Missed

Sets

Cache

Policy A
Policy B

Miss

0: Policy A
1: Policy B

Counter

>0

+1 -1
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

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
Typical memory hierarchies

(a) Memory hierarchy for server

- **CPU**
  - Registers
  - Register reference
  - Size: 1000 bytes
  - Speed: 300 ps

- **L1 Cache**
  - Level 1 Cache reference
  - Size: 64 KB
  - Speed: 1 ns

- **L2 Cache**
  - Level 2 Cache reference
  - Size: 256 KB
  - Speed: 3–10 ns

- **L3 Cache**
  - Level 3 Cache reference
  - Size: 2–4 MB
  - Speed: 10–20 ns

- **Memory**
  - Memory reference
  - Size: 4–16 GB
  - Speed: 50–100 ns

- **I/O bus**
  - Disk storage
  - Disk memory reference
  - Size: 4–16 TB
  - Speed: 5–10 ms

(b) Memory hierarchy for a personal mobile device

- **CPU**
  - Registers
  - Register reference
  - Size: 500 bytes
  - Speed: 500 ps

- **L1 Cache**
  - Level 1 Cache reference
  - Size: 64 KB
  - Speed: 2 ns

- **L2 Cache**
  - Level 2 Cache reference
  - Size: 256 KB
  - Speed: 10–20 ns

- **Memory**
  - Memory reference
  - Size: 256–512 MB
  - Speed: 50–100 ns

- **Storage**
  - FLASH memory reference
  - Size: 4–8 GB
  - Speed: 25–50 us
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>1</th>
<th>1</th>
<th>1</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>100</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>300</td>
<td></td>
<td></td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>204</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>
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
- Nowadays, more general, L4 in Intel Haswell, L3 in IBM Power5
Inclusion Policy

- **Inclusive multilevel cache:**
  - Inner cache holds copies of data in outer cache
  - On miss, line inserted in inner and outer cache; replacement in outer cache invalidates line in inner cache
  - External accesses need only check outer cache
  - Commonly used (e.g., Intel CPUs up to Broadwell)

- **Non-inclusive multilevel caches:**
  - Inner cache may hold data not in outer cache
  - Replacement in outer cache doesn’t invalidate line in inner cache
  - Used in Intel Skylake, ARM

- **Exclusive multilevel caches:**
  - Inner cache and outer cache hold different data
  - Swap lines between inner/outer caches on miss
  - Used in AMD processors

Why choose one type or the other?
Memory Management: 
*From Absolute Addresses to Demand Paging*

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

• The Fifties
  - Absolute Addresses
  - Dynamic address translation

• The Sixties
  - Atlas and Demand Paging
  - Paged memory systems and TLBs

• Modern Virtual Memory Systems
Names for Memory Locations

- **Machine language address**
  - as specified in machine code

- **Virtual address**
  - ISA specifies translation of machine code address into virtual address of program variable (sometimes called *effective* address)

- **Physical address**
  - Operating system specifies mapping of virtual address into name for a physical memory location
Absolute Addresses

EDSAC, early 50’s

Only one program ran at a time, with unrestricted access to entire machine (RAM + I/O devices)

Addresses in a program depended upon where the program was to be loaded in memory

But it was more convenient for programmers to write location-independent subroutines

How could location independence be achieved?

Linker and/or loader modify addresses of subroutines and callers when building a program memory image
Motivation

In the early machines, I/O operations were slow and each word transferred involved the CPU. Higher throughput if CPU and I/O of 2 or more programs were overlapped. *How?*

→ **Multiprogramming**

Location-independent programs

Programming and storage management ease

→ need for a *base register*

Protection

Independent programs should not affect each other inadvertently

→ need for a *bound register*
Simple Base and Bound Translation

- Base and bounds registers are visible/accessible only when the processor is running in **supervisor mode**.

```
Program Address Space

Load X

Effective Address

Base Physical Address

Segment Length

≤

Bounds Violation?

Physical Address

current segment

Main Memory

Bound Register

Base Register
```

Base and bounds registers are visible/accessible only when the processor is running in **supervisor mode**.
Separate Areas for Code and Data

What is an advantage of this separation? (Scheme used on all Cray vector supercomputers prior to X1, 2002)
As users come and go, the storage is “fragmented”. Therefore, at some stage programs have to be moved around to compact the storage.
Paged Memory Systems

Virtual Address Space of User-1

Physical Address Space

0
1
2
3

1
0
3
2
Paged Memory Systems

- Processor-generated address can be interpreted as a pair <page number, offset>

- A page table contains the physical address of the base of each page

Page tables make it possible to store the pages of a program *non-contiguously.*
A Problem in Early Sixties

• There were many applications whose data could not fit in the main memory, e.g., payroll.

  *Paged memory system reduced fragmentation but still required the whole program to be resident in the main memory*

• Programmers moved the data back and forth from the secondary store by *overlaying* it repeatedly on the primary store.

  *tricky programming!*
Manual Overlays

- Assume an instruction can address all the storage on the drum

- **Method 1**: programmer keeps track of addresses in the main memory and initiates an I/O transfer when required

- **Method 2**: automatic initiation of I/O transfers by software address translation

  *Brooker’s interpretive coding, 1960*

Problems?

- Method 1: Difficult, error prone
- Method 2: Inefficient
Demand Paging in Atlas (1962)

“A page from secondary storage is brought into the primary storage whenever it is (implicitly) demanded by the processor.”

Tom Kilburn

Primary memory as a cache for secondary memory

User sees the storage size of the secondary storage, since data transfer happens automatically
Hardware Organization of Atlas

Effective Address

Initial Address Decode

16 ROM pages
0.4 ~ 1 \( \mu \)sec

2 subsidiary pages
1.4 \( \mu \)sec

Main
32 pages
1.4 \( \mu \)sec

Drum (4)
192 pages

8 drum drives
88 sec/word

48-bit words
512-word pages

1 Page Address Register (PAR)
per page frame in main memory

<effective PN, status>

Compare the effective page address against all 32 PARs

match \( \Rightarrow \) normal access

no match \( \Rightarrow \) page fault

save the state of the partially executed instruction
Atlas Demand Paging Scheme

• On a page fault:
  – Input transfer into a free page is initiated
  – The Page Address Register (PAR) is updated
  – If no free page is left, a page is selected to be replaced (based on usage)
  – The replaced page is written on the drum
    • to minimize the drum latency effect, the first empty page on the drum was selected
  – The page table is updated to point to the new location of the page on the drum
Caching vs. Demand Paging

**Caching**
- cache entry
- cache block (~32 bytes)
- cache miss rate (1% to 20%)
- cache hit (~1 cycle)
- cache miss (~100 cycles)
- a miss is handled in *hardware*

**Demand paging**
- page frame
- page (~4K bytes)
- page miss rate (<0.001%)
- page hit (~100 cycles)
- page miss (~5M cycles)
- a miss is handled mostly in *software*
Modern Virtual Memory Systems

*Illusion of a large, private, uniform store*

**Protection & Privacy**
several users, each with their private address space and one or more shared address spaces
page table $\equiv$ name space

**Demand Paging**
Provides the ability to run programs larger than the primary memory
Hides differences in machine configurations

*The price is address translation on each memory reference*
Next lecture:
Modern Virtual Memory Systems