Lecture 10: Caches
The Cache Hierarchy
Accessing the Cache

Direct-mapped cache: each address maps to a unique address.

8-byte words

Sets

Data array

Offset

Byte address

101000

8 words: 3 index bits
The Tag Array

Direct-mapped cache: each address maps to a unique address
Increasing Line Size

A large cache line size → smaller tag array, fewer misses because of spatial locality

32-byte cache line size or block size
Associativity

Set associativity $\rightarrow$ fewer conflicts; wasted power because multiple data and tags are read.

Byte address

10100000

Tag array

Tag

Compare

Way-1

Way-2

Data array
Example

- 32 KB 4-way set-associative data cache array with 32 byte line sizes
- How many sets?
- How many index bits, offset bits, tag bits?
- How large is the tag array?
Types of Cache Misses

• Compulsory misses: happens the first time a memory word is accessed – the misses for an infinite cache

• Capacity misses: happens because the program touched many other words before re-touching the same word – the misses for a fully-associative cache

• Conflict misses: happens because two words map to the same location in the cache – the misses generated while moving from a fully-associative to a direct-mapped cache

• Sidenote: can a fully-associative cache have more misses than a direct-mapped cache of the same size?
Reducing Miss Rate

• Large block size – reduces compulsory misses, reduces miss penalty in case of spatial locality – increases traffic between different levels, space waste, and conflict misses

• Large cache – reduces capacity/conflict misses – access time penalty

• High associativity – reduces conflict misses – rule of thumb: 2-way cache of capacity N/2 has the same miss rate as 1-way cache of capacity N – more energy
More Cache Basics

- L1 caches are split as instruction and data; L2 and L3 are unified

- The L1/L2 hierarchy can be inclusive, exclusive, or non-inclusive

- On a write, you can do write-allocate or write-no-allocate

- On a write, you can do writeback or write-through; write-back reduces traffic, write-through simplifies coherence

- Reads get higher priority; writes are usually buffered

- L1 does parallel tag/data access; L2/L3 does serial tag/data
Techniques to Reduce Cache Misses

- Victim caches
- Better replacement policies – pseudo-LRU, NRU, DRRIP
- Cache compression
Victim Caches

• A direct-mapped cache suffers from misses because multiple pieces of data map to the same location

• The processor often tries to access data that it recently discarded – all discards are placed in a small victim cache (4 or 8 entries) – the victim cache is checked before going to L2

• Can be viewed as additional associativity for a few sets that tend to have the most conflicts
Replacement Policies

• Pseudo-LRU: maintain a tree and keep track of which side of the tree was touched more recently; simple bit ops

• NRU: every block in a set has a bit; the bit is made zero when the block is touched; if all are zero, make all one; a block with bit set to 1 is evicted

• DRRIP: use multiple (say, 3) NRU bits; incoming blocks are set to a high number (say 6), so they are close to being evicted; similar to placing an incoming block near the head of the LRU list instead of near the tail
Tolerating Miss Penalty

- Out of order execution: can do other useful work while waiting for the miss – can have multiple cache misses -- cache controller has to keep track of multiple outstanding misses (non-blocking cache)

- Hardware and software prefetching into prefetch buffers – aggressive prefetching can increase contention for buses
Stream Buffers

- Simplest form of prefetch: on every miss, bring in multiple cache lines
- When you read the top of the queue, bring in the next line

![Diagram of L1 stream buffer with sequential lines](image-url)
Stride-Based Prefetching

- For each load, keep track of the last address accessed by the load and a possibly consistent stride
- FSM detects consistent stride and issues prefetches
Prefetching

• Hardware prefetching can be employed for any of the cache levels

• It can introduce cache pollution – prefetched data is often placed in a separate prefetch buffer to avoid pollution – this buffer must be looked up in parallel with the cache access

• Aggressive prefetching increases “coverage”, but leads to a reduction in “accuracy” \(\Rightarrow\) wasted memory bandwidth

• Prefetches must be timely: they must be issued sufficiently in advance to hide the latency, but not too early (to avoid pollution and eviction before use)
Intel Montecito Cache

Two cores, each with a private 12 MB L3 cache and 1 MB L2

Naffziger et al., Journal of Solid-State Circuits, 2006
Shared Vs. Private Caches in Multi-Core

- What are the pros/cons to a shared L2 cache?
Shared Vs. Private Caches in Multi-Core

• Advantages of a shared cache:
  ▪ Space is dynamically allocated among cores
  ▪ No waste of space because of replication
  ▪ Potentially faster cache coherence (and easier to locate data on a miss)

• Advantages of a private cache:
  ▪ small L2 → faster access time
  ▪ private bus to L2 → less contention
UCA and NUCA

• The small-sized caches so far have all been uniform cache access: the latency for any access is a constant, no matter where data is found

• For a large multi-megabyte cache, it is expensive to limit access time by the worst case delay: hence, non-uniform cache architecture
Large NUCA

Issues to be addressed for Non-Uniform Cache Access:

- Mapping
- Migration
- Search
- Replication
Shared NUCA Cache

A single tile composed of a core, L1 caches, and a bank (slice) of the shared L2 cache.

The cache controller forwards address requests to the appropriate L2 bank and handles coherence operations.
Virtual Memory

- Processes deal with virtual memory – they have the illusion that a very large address space is available to them.

- There is only a limited amount of physical memory that is shared by all processes – a process places part of its virtual memory in this physical memory and the rest is stored on disk.

- Thanks to locality, disk access is likely to be uncommon.

- The hardware ensures that one process cannot access the memory of a different process.
Virtual Memory and Page Tables
Address Translation

- The virtual and physical memory are broken up into pages

8KB page size

Virtual address

- virtual page number
- page offset

Translated to phys page number

Physical address

- physical page number
- page offset

Physical memory
Memory Hierarchy Properties

- A virtual memory page can be placed anywhere in physical memory (fully-associative)

- Replacement is usually LRU (since the miss penalty is huge, we can invest some effort to minimize misses)

- A page table (indexed by virtual page number) is used for translating virtual to physical page number

- The memory-disk hierarchy can be either inclusive or exclusive and the write policy is writeback
Since the number of pages is very high, the page table capacity is too large to fit on chip

A translation lookaside buffer (TLB) caches the virtual to physical page number translation for recent accesses

A TLB miss requires us to access the page table, which may not even be found in the cache – two expensive memory look-ups to access one word of data!

A large page size can increase the coverage of the TLB and reduce the capacity of the page table, but also increases memory waste
TLB and Cache

- Is the cache indexed with virtual or physical address?
  - To index with a physical address, we will have to first look up the TLB, then the cache \(\Rightarrow\) longer access time
  - Multiple virtual addresses can map to the same physical address – can we ensure that these different virtual addresses will map to the same location in cache? Else, there will be two different copies of the same physical memory word

- Does the tag array store virtual or physical addresses?
  - Since multiple virtual addresses can map to the same physical address, a virtual tag comparison can flag a miss even if the correct physical memory word is present
TLB and Cache
Virtually Indexed Caches

- 24-bit virtual address, 4KB page size → 12 bits offset and 12 bits virtual page number
- To handle the example below, the cache must be designed to use only 12 index bits – for example, make the 64KB cache 16-way
- Page coloring can ensure that some bits of virtual and physical address match

![Diagram showing virtually indexed cache and page in physical memory]
Thank you!
Problem 1

- Memory access time: Assume a program that has cache access times of 1-cyc (L1), 10-cyc (L2), 30-cyc (L3), and 300-cyc (memory), and MPKIs of 20 (L1), 10 (L2), and 5 (L3). Should you get rid of the L3?
Problem 1

- Memory access time: Assume a program that has cache access times of 1-cyc (L1), 10-cyc (L2), 30-cyc (L3), and 300-cyc (memory), and MPKIs of 20 (L1), 10 (L2), and 5 (L3). Should you get rid of the L3?

With L3: $1000 + 10 \times 20 + 30 \times 10 + 300 \times 5 = 3000$
Without L3: $1000 + 10 \times 20 + 10 \times 300 = 4200$