Advanced Caches

Introduction

1. Cover multilevel caches and various cache optimizations
   - Important for good overall performance and energy efficiency

Improving Cache Performance

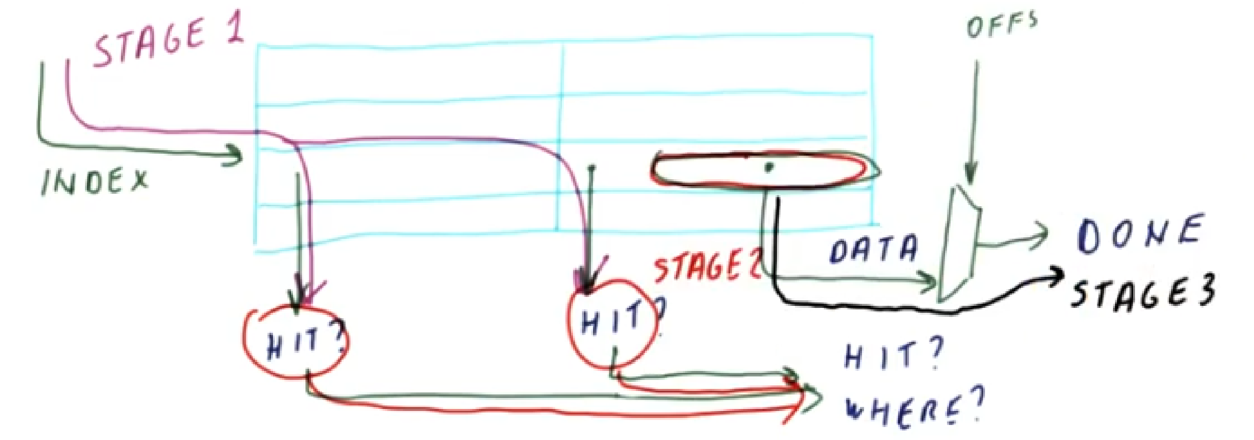
1. AMAT: Average memory access time
   - AMAT = Hit time + Miss rate * miss penalty
2. Methods for improving cache performance
   - Reduce hit time
   - Reduce miss rate
   - Reduce miss penalty

Reduce Hit Time

1. Reduce cache size
   - Bad for miss rate (might not improve AMAT)
   - Need to be careful to balance
2. Reduce cache associativity
   - Bad for miss rate because there will be more conflicts
3. Overlap cache hit with another hit
4. Overlap cache hit with TLB hit
5. Optimize lookup for common case
6. Maintain replacement state more quickly

Pipelined Caches

1. Multiple cycles to access
   - Access comes in cycle N (hit)
   - Second access comes in cycle N+1 (hit)
     - Has to wait if cache isn't pipelined
   - Hit time = Actual hit + wait time
2. Pipelining a cache
   - Stage 1: Reading tags from cache
   - Stage 2: Determining hits and beginning data read
   - Stage 3: Finishing data read and getting data
3. Level 1 caches are typically pipelined



Pipelined Caches

TLB And Cache Hit

1. Use VA to index into TLB to get frame number, combine with page offset to get physical address
    - Use this to access cache and get data
    - TLB and cache will both require one cycle
    - Overall cache latency is TLB hit latency and cache hit latency
2. Cache that is indexed using physical address is called:
    - Physically accessed cache
    - Physical cache
    - Physically indexed-physically tagged cache (PIPT)

Virtually Accessed Cache

1. If a cache is accessed using the virtual address, it's considered a virtually accessed cache
    - On a cache miss, we need to get the physical address to bring data into the cache
    - On a cache hit, we can get the data without accessing the TLB at all
2. Advantages:
    - Hit time = Cache hit time
    - No TLB access on cache hit (only in theory)
3. Disadvantages:
    - TLB contains permissions (rwx) that we need to determine access
        - This means we must still access the TLB, even on a cache hit
    - On a context switch, we must flush the cache
        - Different processes will have different VA->PA mappings
        - Processes are switch roughly once per millisecond

Virtually Indexed Physically Tagged

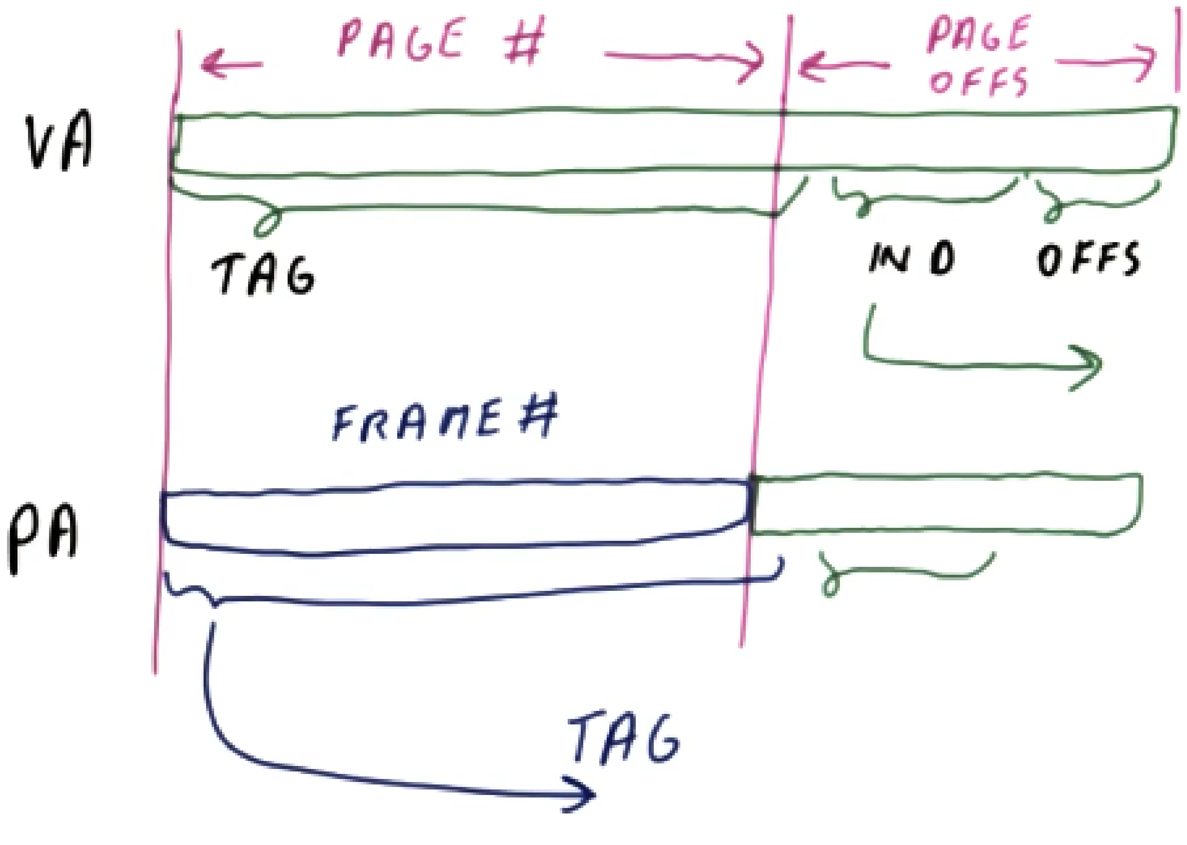
1. Tries to combine the advantages of the two types of caches
2. Uses bits from the virtual address to access the cache and bits from the physical address to check the tag
    - Cache and TLB access are proceeding in parallel
3. Advantages:
    - Hit time = Cache hit time (like VIVT)
    - Don't need to flush on a context switch (like PIPT)
        - Tags from different processes are guaranteed to be unique
    - Aliasing is not an issue if the cache is small enough

Aliasing in Virtually Accessed Caches

1. Aliasing occurs when two virtual addresses map to the same physical address
   - Then, it's possible that one cache entry is updated while the other isn't
   - Require some way to maintain coherence in this case (update or invalidate)

VIPT Cache Aliasing

1. The virtual address is split into tag, index, and offset bits for accessing the cache and page number/page offset bits for accessing the physical address
   - If the cache is small and all of the index bits are contained within the page offset bits, there is no aliasing
     - These bits would be the same if the cache was physically indexed
2. Consider the following cache:
   - 4 kB page -> 12-bit page offset
   - 32 B block -> 5-bit block offset
   - Index needs to fit in the 12 bits of page offset
     - 12 - 5 = 7, so can only have a 7-bit index -> 128 sets



Aliasing in a VIPT Cache

VIPT Aliasing Avoidance Quiz

1. Consider the following cache:
   - 4-way set-associative
   - 16-byte block size

   - 8 kB page size
2. If we want no aliasing, what is the maximum size of the cache?
   - 8 kB page -> 13-bit page offset
   - 16 B block -> 4-bit block offset
   - 2 ^ 9 * 2 ^ 4 * 2 ^ 2 = 2 ^ 15 = 32 kB
     - Bytes per block, blocks per set, number of sets
3. The only way to avoid aliasing in a cache while making it larger is to increase associativity

Real VIPT Caches

1. Cache size must be less than or equal to the associativity * page size
   - Pentium 4: 4 way SA * 4 kB -> L1 is 16 kB
   - Core 2: 8 way SA * 4 kB -> L1 is 32 kB
   - Sandy Bridge: 8 way SA * 4 kB -> L1 is 32 kB
   - Haswell: 8 way SA * 4 kB -> L1 is 32 kB
   - Skylake: 16-way SA * 4 kB -> L1 is 64 kB

Associativity and Hit Time

1. High associativity
   - Fewer conflicts -> Lower miss rate :)
   - Larger VIPT caches -> Lower miss rate :)
   - Slower hits :(
2. Direct mapped
   - Miss rate is increased, but hit time is reduced
3. Want the hit time of a direct mapped cache with the conflict reduction of a highly associative cache
   - Cheat on associativity

Way Prediction

1. Start with a set-associative cache (low miss rate)
   - Guess which line in the set is most likely to hit (reduces hit time)
     - First, only check the tag that is most likely
     - If it misses, do a normal set-associative check

Way Prediction Performance

1. First, try to access what looks like a smaller, direct-mapped cache
   - Then, try the entire set-associative cache
2. AMAT calculations
   - 2 + 0.1 * 20 = 4
   - 1 + 0.3 * 20 = 7
   - 1 * 0.7 + 2 * 0.3 + 0.1 * 20 = 3.3

	32kB, 8-way SA	4 kB DM	32 kB 8-way SA Way Pred
Hit Rate	90%	70%	90%
Hit Latency	2	1	1 or 2
Miss Penalty	20	20	20
AMAT	4	7	3.3

Way Prediction Quiz

1. We can use way prediction in
   - Fully associative (yes)

- 8-way set associative (yes)
- 2-way set associative (yes)
- Direct-mapped (no)

Replacement Policy and Hit Time

1. Random: Nothing to update on cache hit
    - Decreases hit time but increases hit rate
2. LRU: Update lots of counters on hit
    - Decreases miss rate but increases hit time
    - Even if we access the most recently used block, we still have to check all of the other counters

NMRU Replacement

1. Not Most Recently Used
    - Tries to approximate performance of LRU with less activity on hits
    - Track which block in set is MRU
    - On replacement, pick any non-MRU block
2. N-way set-associative tracking of MRU
    - One MRU pointer/set (vs N LRU counters)
3. Hit rate is slightly lower than true LRU, but might make hit time go 2 -> 1
    - Most frequently accessed thing will be saved, but it's possible that the second MRU is selected
    - Want an algorithm that tracks more state

PLRU Replacement

1. Pseudo Most Recently Used
    - Track one bit per line in the set
    - Every time a line is accessed, set its counter to 1
        - Keep doing while there is at least one counter is 0
    - When the final bit is set to 1, zero out all other counters
        - Now, we only know the MRU block
2. When one bit is set, we're implementing the NRMU policy
3. When between one and all but one bits are set, we're in between NMRU/LRU
4. When all but one bits are set, we're implementing the LRU policy
5. Both NRMU and PLRU have much less activity on a hit, which helps with power and reduces hit time

NMRU Quiz

1. Fully associative cache with 4 lines
    - NMRU replacement
    - Start empty
2. Processor accesses blocks in the following order:
    - A, A, B, A, C, A, D, A, E, A, A, A, A, B
3. What's the smallest number of cache misses we will have?
    - M, H, M, H, M, H, M, H, M, H, H, H, H, H
    - Five (first access of A B C D E)
4. What's the largest number of cache misses we will have?
    - M, H, M, H, M, H, M, H, M, H, H, H, H, M
    - Six (if B is replaced by E)
5. Don't start evicting until all lines are full

Reducing the Miss Rate

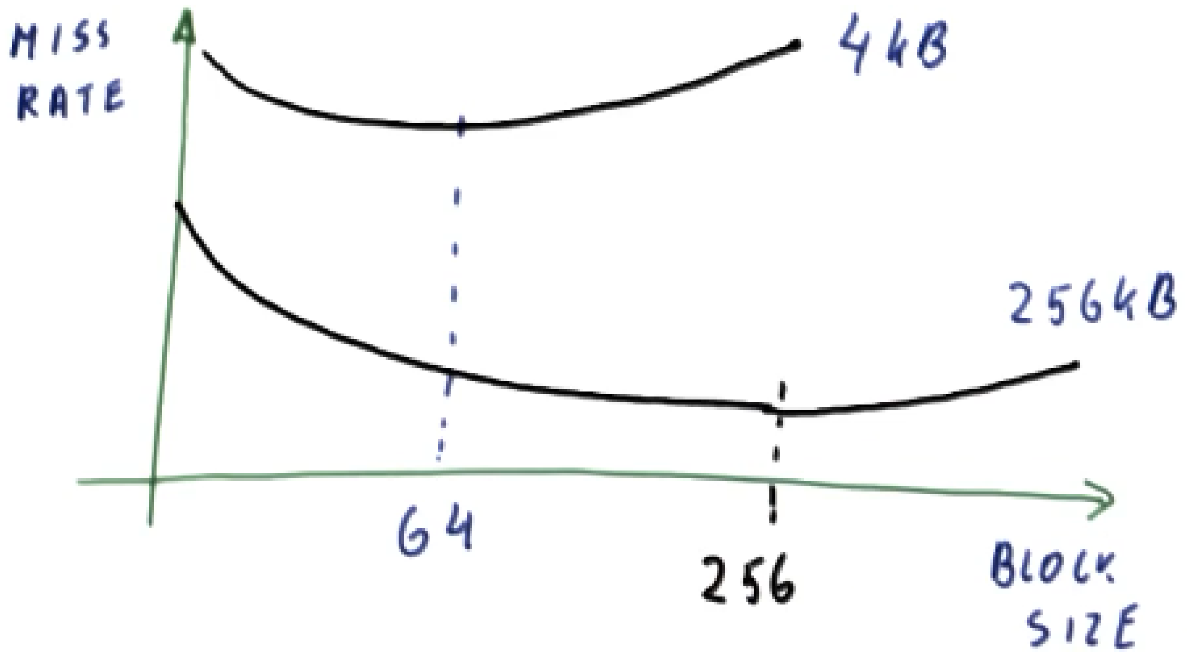
1. What are the causes of misses?

- Compulsory misses: First time a block is accessed
  - Would be a miss even in infinite cache
- Capacity misses: Block evicted because of limited cache size
  - Would be a miss even in fully-associative cache of that size
- Conflict misses: Block evicted because of limited associativity
  - Would not be a miss in a fully-associative caches

2. Ways to improve miss rate
   - Larger cache reduces capacity misses
   - More associative cache reduces conflict misses
   - Better replacement algorithm reduces conflict misses
   - All of these also affect the hit time

Larger Cache Blocks

1. More words are brought in on a miss
   - Subsequent accesses won't be misses
   - Reduces miss rate when spatial locality is good
   - Increases miss rate when spatial locality is poor
2. Larger caches allow for larger blocks because the larger size allows for more junk to be accomodated (due to poor locality)



Miss Rate as a Function of Block Size

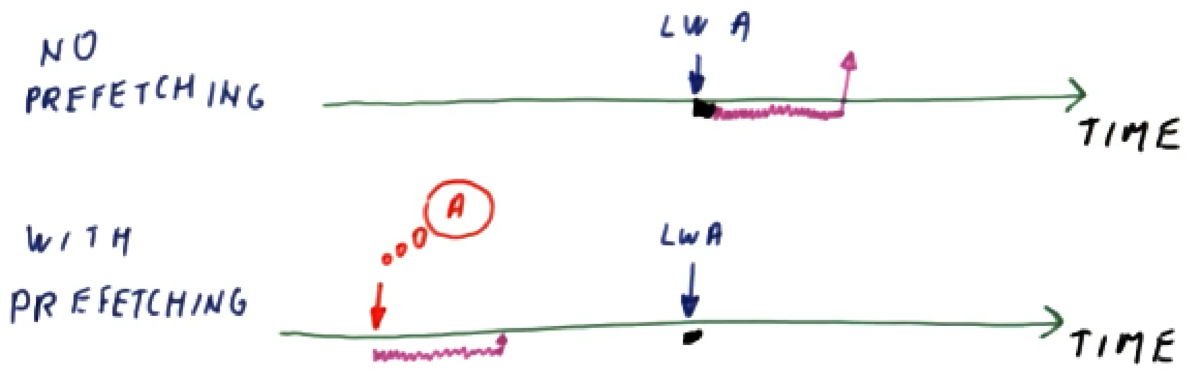
Miss Rate Quiz

1. When the block size is increased, which types of misses are reduced?
   - Compulsory (yes)
     - Compulsory misses occur when a block is accessed for the first time
     - Larger blocks mean there are fewer blocks
   - Capacity (yes)

    - Once the cache is full, the number of misses is equal to the number of blocks; larger blocks means fewer blocks, so fewer misses
- Conflict (yes)
    - When a block is evicted, the new block can capture more data if the block is larger

Prefetching

1. Guess which blocks will be accessed soon and bring them into cache ahead of time
2. Good guess -> Eliminate a miss
3. Bad guess -> Cache pollution -> Didn't eliminate miss and may have caused another miss



Prefetching

Prefetch Instructions

1. Original program:

```
for(int i = 0; i < 100000; i++
{
    sum += a[i];
}
```

2. Program with prefetching:

```
for(int i = 0; i < 100000; i++
{
    prefetch a[i+pdist];
    sum += a[i];
}
```

3. What's the right pdist to use?
    - If pdist is too small, access happens before data arrives (though slightly less expensive)
    - If pdist is too large, the prefetched data might be evicted before it's used due to subsequent accesses
    - The correct value for pdist changes as the underlying hardware changes

Prefetch Instructions Quiz

1. Consider the following program:

```
for(int i = 0; i < 1000; i++)
{
    for(int j = 0; j < 1000; j++)
    {
        a[i] = a[i] + b[i][j];
    }
}
```

2. Assume the following:
   - 8-byte elements
   - Cache size is 16 kB
   - Fully associative, LRU replacement
   - 10 cycles if no misses
   - Miss penalty (memory latency) is 200 cycles
3. If we insert a prefetch instruction for array a, what should pdist be?
   - pdist should be 1 so the next element is brought in
   - If the entire cache was filled by the inner loop, we shouldn't prefetch at all
4. If we insert a prefetch instruction for array b, what should pdist be?
   - 200 / 10 = 20 iterations

Hardware Prefetching

1. No change to the program
2. Hardware tries to guess what will be accessed soon
3. Popular modern prefetchers
   - Stream buffer: Sequential accesses
   - Stride prefetcher: See if accesses are a fixed distance apart
   - Correlating prefetcher: Has a table to remember that after A is accessed, B is typically accessed
     - Then, when A is accessed again, B is prefetched
     - Good for linked lists (not sequential or constant stride)

Loop Interchange

1. C arrays are arranged in row-major order in memory
   - If the elements are accessed in column-major order, we lose out on spatial locality optimizations
   - A good compiler will detect that the order of the loops doesn't match the layout in memory
   - Compiler must prove that the modified code is equivalent by showing there are no dependencies
2. Compiler can rearrage the loops in the first program into the order shown in the second program

```
for(int i = 0; i < 1000; i++)
{
    for(int j = 0; j < 1000; j++)
    {
        a[j][i] = 0;
    }
}
```

```
for(int j = 0; j < 1000; j++)
{
    for(int i = 0; i < 1000; i++)
    {
        a[j][i] = 0;
    }
}
```

Overlap Misses

1. Blocking cache: One cache miss prevents further memory accesses from future cache misses
2. We can reduce the miss penalty by using a non-blocking cache

    - Hit under miss: Cache hit while the cache is already processing a cache miss
    - Miss under miss: Cache miss while the cache is already processing a cache miss
3. This reduces the penalty of a cache miss
4. This is called memory-level parallelism

Miss Under Miss Support in Caches

1. Miss status handling registers (MSHRs)
    - Keep information about ongoing misses
    - Checks MSHRs to see if any match
        - No match: Allocate an MSHR and remember which instruction to wake up
        - Match: Would be a hit if the cache was blocking, but data is already being retrieved (half-miss)
    - On a match, we add the instruction to the MSHR
        - When the data comes back, wake up all instructions subscribing to it
2. How many MSHRs do we want?
    - Huge benefit even to having only two MSHRs
    - 2 is good, 4 is better, 16-32 is even better

Miss Under Miss Quiz

1. What kind of application gets no benefit from miss-under-miss support?
    - Application that always hits in the cache (yes)
    - Application that has a miss every 1000 instructions (yes)
    - Application that has a miss every 10 instructions (no)
    - Application that has a miss every 2 instructions (no)

Cache Hierarchies

1. A cache hierarchy also reduces the miss penalty
    - Miss in L1 cache goes to another cache
    - L1 miss penalty != memory latency
    - L1 miss penalty = L2 hit time + L2 miss rate * L2 miss penalty

AMAT With Cache Hierarchies

1. AMAT = L1 hit time + L1 miss rate * L1 miss penalty
2. L1 miss penality = L2 hit time = L2 miss rate * L2 miss penalty
3. LN miss penalty = Main memory latency
    - LLC: Last level cache

L1 vs L2

1. Which of these statements are true?
    - L1 capacty < L2 capacity (true)
    - L1 latency < L2 latency (true)
    - L1 # of accesses < L2 # of accesses (false)
    - L1 associativity == L2 associativity (false)
        - L1 cache needs low hit latency
        - L2 cache can be bigger with higher hit latency so it can have a higher associativity

Multilevel Cache Performance

1. Assume memory latency is 100 cycles
2. AMAT calculations
    - 2 + 0.1 * 100 = 12

- 10 + 0.025 * 100 = 12.5
- 2 + 0.1 * (10 + 0.25 * 100) = 5.5

3. A large and slow cache is insufficient

	16 kB	128 kB	No cache	L1 = 16 kB, L2 = 128 kB
Hit Time	2	10	100	2 for L1, 10 + 2 for L2
Hit Rate	90%	97.5%	100%	90% L1, 75% for L2
AMAT	12	12.5	100	5.5

Hit Rate

1. In the previous example, the hit rate for the L2 cache is 75% because the first 90% are handled by the L1 cache
    - This is known as the local hit rate of the cache (hit rate the cache actually observes)
    - The local hit rate is lower in L2 because so many of the accesses can be handled by the L1 cache
2. We typically consider the global hit rate because it describes the overall performance of the system

Global vs Local Hit Rate

1. Global hit rate: 1 - Global miss rate
2. Global miss rate: # of misses in this cache / # of all memory references
3. Local hit rate: # of hits / # of accesses to this cache
4. Misses per 1000 instructions (MPKI) is another popular metric

Global vs Local Hit Rate Quiz

1. L1 cache has 90% hit rate
    - Local miss rate is 10%
    - Global miss rate is 10%
2. L2 cache hits for 50% of L1 misses
    - Local miss rate is 50%
    - Global miss rate is 5%

Inclusion Property

1. Assume a block is in L1 cache
    - May or may not be in L2 (neither inclusion nor exclusion)
    - Has to also be in L2 (inclusion)
    - Cannot also be in L2 (exclusion)
2. Unless we explicitly enforce inclusion or exclusion, we end up with the first option
    - Typically maintain an additional inclusion bit
3. Inclusion is useful for simplifying how cache coherence is handled
    - If the L1 cache is write-through, inclusion ensures that a write that is a L1 hit will actually happen in L2 (not be an L2 miss)

Inclusion Quiz

1. L1, L2 maintain inclusion property
    - Dirty block replaced from L1 -> write back
    - Can the write-back be an L2 hit?
        - Yes
    - Can the write-back be an L2 miss?
        - No

2. If there is no attempt to maintain inclusion
    - Can the write-back be an L2 hit?
        - Yes
    - Can the write-back be an L2 miss?
        - Yes

Conclusion

1. Modern processors use multi-level processors and other cache tricks to compensate for main memory being so slow
    - Why is main memory slow?