Measuring and Improving Cache Performance

Two different techniques for improving cache performance.

1. Focuses on reducing the miss rate by reducing the probability that two different memory blocks will contend for the same cache location

2. Reduce the miss penalty by adding an additional level to the hierarchy - MULTILEVEL CACHING

CPU time can be divided into

- clock cycles that the CPU spends executing the program
- clock cycles that the CPU spends waiting for the memory system.

Normally, we assume that the costs of cache accesses that are hits are part of the normal CPU execution cycles. Thus,

$$\text{CPU time} = (\text{CPU execution clock cycles} + \text{Memory-stall clock cycles}) \times \text{Clock cycle time}$$

Measuring and Improving Cache Performance

$$\text{CPU time} = (\text{CPU execution clock cycles} + \text{Memory-stall clock cycles}) \times \text{Clock cycle time}$$

$$\text{Memory-stall clock cycles} = (\text{Read-stall cycles} + \text{Write-stall cycles})$$

$$\text{Read-stall cycles} = \frac{\text{Reads}}{\text{Program}} \times \text{Read Miss rate} \times \text{Read miss penalty}$$

Writes are more complicated. For a write-through scheme, we have two sources of stalls: write misses, which usually require that we fetch the block before continuing the write, and write buffer stalls, which occur when the write buffer is full when a write occurs.

Thus, the cycles stalled for writes equals the sum of these two:

$$\text{Write-stall cycles} = \left(\frac{\text{Writes}}{\text{Program}} \times \text{Write Miss rate} \times \text{Write miss penalty}\right) + \text{Write buffer stalls}$$

Measuring and Improving Cache Performance

In most write-through cache organizations, the read and write miss penalties are the same (the time to fetch the block from memory). If we assume that the write buffer stalls are negligible, we can combine the reads and writes by using a single miss rate and the miss penalty:

$$\text{Memory-stall clock cycles} = \left(\frac{\text{Memory accesses}}{\text{Program}} \times \text{Miss rate} \times \text{Miss penalty}\right)$$

Or

$$\text{Memory-stall clock cycles} = \left(\frac{\text{Memory accesses}}{\text{Program}} \times \frac{\text{M}\mathit{isses}}{\text{Instruction}} \times \text{Miss penalty}\right)$$

Measuring and Improving Cache Performance

Calculating Cache Performance

Assume the miss rate of an instruction cache is 2% and the miss rate of the data cache is 4%. If a processor has a CPI of 2 without any memory stalls and the miss penalty is 100 cycles for all misses, determine how much faster a processor would run with a perfect cache that never missed. Assume the frequency of all loads and stores is 36%.

The number of memory miss cycles for instructions in terms of the Instruction count (I) is

$$\text{Instruction Miss cycles} = I \times 2\% \times 100 = 2.00 \times I$$

As the frequency of all loads and stores is 36%, we can find the number of memory miss cycles for data references:

$$\text{Data miss cycles} = I \times 36\% \times 4\% \times 100 = 1.44 \times I$$

Measuring and Improving Cache Performance

Calculating Cache Performance

The total number of memory-stall cycles is 2.00 I + 1.44 I = 3.44 I. This is more than three cycles of memory stall per instruction.

Accordingly, the total CPI including memory stalls is 2+ 3.44 = 5.44. Since there is no change in instruction count or clock rate, the ratio of the CPU execution times is

$$\frac{\text{CPU time with stalls}}{\text{CPU time with perfect cache}} = \frac{\text{I} \times \text{CPI}_{\text{stall}} \times \text{Clock cycle}}{\text{I} \times \text{CPI}_{\text{perfect}} \times \text{Clock cycle}}$$

$$= \frac{\text{CPI}_{\text{stall}}}{\text{CPI}_{\text{perfect}}} = \frac{5.44}{2} = 2.72$$

Performance with the perfect cache is better by 2.72

Measuring and Improving Cache Performance

What happens if the processor is made faster, but the memory system is not?

Suppose we speed-up the computer in the previous example by reducing its CPI from 2 to 1 without changing the clock rate, which might be done with an improved pipeline.

The system with cache misses would then have a CPI of $1 + 3.44 = 4.44$, and the system with the perfect cache would be

$$\frac{4.44}{1} = 4.44 \text{ times as fast}$$

The amount of execution time spent on memory stalls would have risen from

$$\frac{3.44}{5.44} = 63\% \quad \text{to} \quad \frac{3.44}{4.44} = 77\%$$

Similarly, increasing the clock rate without changing the memory system also increases the performance lost due to cache misses.

Measuring and Improving Cache Performance

Till now, we assumed that the hit time is not a factor in determining cache performance.

If the hit time increases (cache size increases), the total time to access a word from the memory system will increase, possibly causing an increase in the processor cycle time.

An increase in hit time likely adds another stage to the pipeline, since it may take multiple cycles for a cache hit.

To capture the fact that the time to access data for both hits and misses affects performance, designers sometime use average memory access time (AMAT) as a way to examine alternative cache designs.

Average memory access time is the average time to access memory considering both hits and misses and the frequency of different accesses; it is equal to the following:

$$\text{AMAT} = \text{Time for a hit} + \text{Miss rate} \times \text{Miss penalty}$$

Measuring and Improving Cache Performance

Calculating Average Memory Access Time

Find the AMAT for a processor with a 1 ns clock cycle time, a miss penalty of 20 clock cycles, a miss rate of 0.05 misses per instruction, and a cache access time (including hit detection) of 1 clock cycle. Assume that the read and write miss penalties are the same and ignore other write stalls.

The average memory access time per instruction is

AMAT = Time for a hit + Miss rate x Miss penalty

= 1 + 0.05 x 20

= 2 clock cycles

or 2 ns.

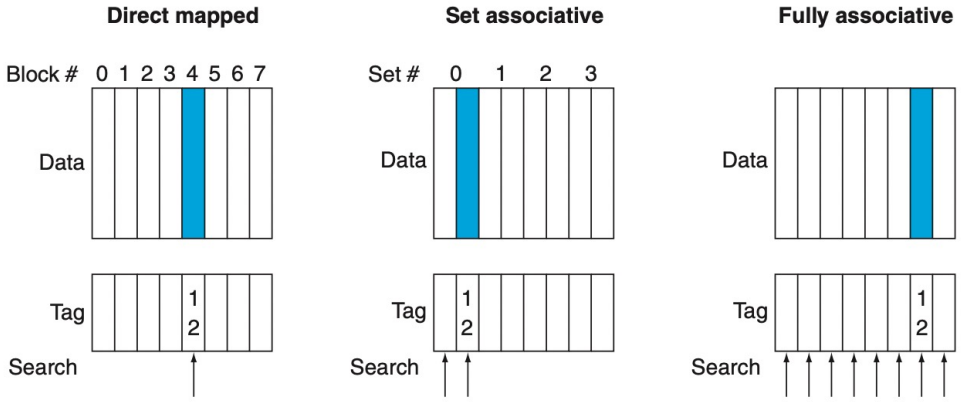
Reducing Cache Misses by More Flexible Placement of Blocks

Alternative cache organizations that decrease miss rate but may sometimes increase hit time.

Direct-Mapped : A block can go in exactly one place in the cache.

Fully Associative : A block can be placed in any location in the cache.

- To find a given block, all the entries in the cache must be searched.
- Search is performed in parallel with a comparator associated with each cache entry. Increase the hardware cost, practical only for caches with small numbers of blocks.



Reducing Cache Misses by More Flexible Placement of Blocks

Set-associative: Design between direct mapped and fully associative is called set associative.

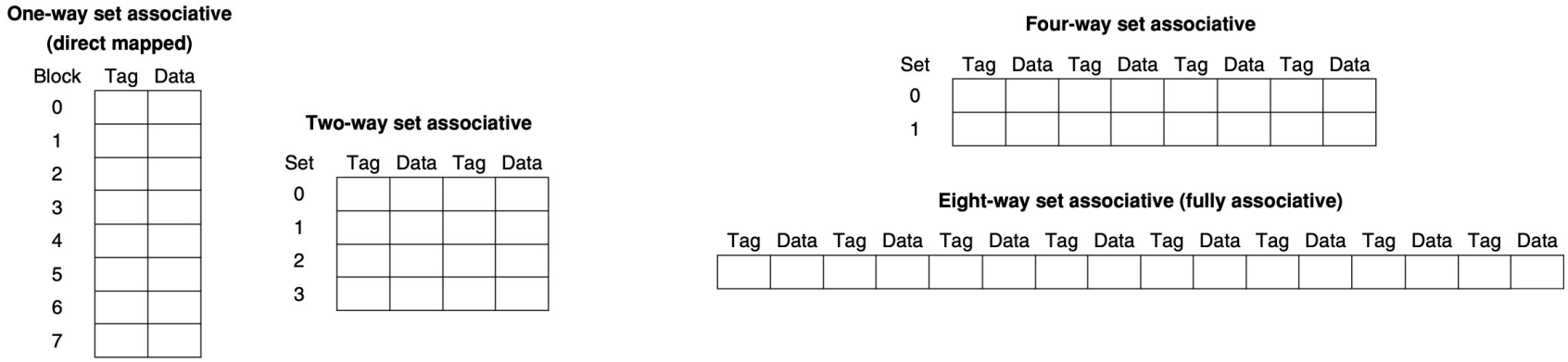
- In a set-associative cache, there are a fixed number of locations where each block can be placed.
- A set-associative cache with *n* locations for a block is called an n-way set-associative cache.
- An n-way set-associative cache consists of a number of sets, each of which consists of n blocks.
- Each block in the memory maps to a unique set in the cache given by the index field, and a block can be placed in any element of that set.
- Since the block may be placed in any element of the set, all the tags of all the elements of the set must be searched.

Reducing Cache Misses by More Flexible Placement of Blocks

Remember that in a direct-mapped cache, the position of a memory block is given by
(Block number) modulo (Number of blocks in the cache)

In a set-associative cache, the set containing a memory block is given by
(Block number) modulo (Number of sets in the cache)

The advantage of increasing the degree of associativity is that it usually decreases the miss rate.
The main disadvantage, is a potential increase in the hit time.



Reducing Cache Misses by More Flexible Placement of Blocks

Misses and Associativity in Caches

Assume there are three small caches, each consisting of four one-word blocks. One cache is fully associative, a second is two-way set-associative, and the third is direct-mapped. Find the number of misses for each cache organization given the following sequence of block addresses: 0, 8, 0, 6, and 8.

Direct-Mapped

Block address	Cache block
0	(0 modulo 4) = 0
6	(6 modulo 4) = 2
8	(8 modulo 4) = 0

Address of memory block accessed	Hit or miss	Contents of cache blocks after reference			
		0	1	2	3
0	miss	Memory[0]			
8	miss	Memory[8]			
0	miss	Memory[0]			
6	miss	Memory[0]		Memory[6]	
8	miss	Memory[8]		Memory[6]	

Reducing Cache Misses by More Flexible Placement of Blocks

Misses and Associativity in Caches

Assume there are three small caches, each consisting of four one-word blocks. One cache is fully associative, a second is two-way set-associative, and the third is direct-mapped. Find the number of misses for each cache organization given the following sequence of block addresses: 0, 8, 0, 6, and 8.

Set-associative – 2 sets with indices 0 and 1

Block address	Cache set
0	(0 modulo 2) = 0
6	(6 modulo 2) = 0
8	(8 modulo 2) = 0

Address of memory block accessed	Hit or miss	Contents of cache blocks after reference			
		Set 0	Set 0	Set 1	Set 1
0	miss	Memory[0]			
8	miss	Memory[0]	Memory[8]		
0	hit	Memory[0]	Memory[8]		
6	miss	Memory[0]	Memory[6]		
8	miss	Memory[8]	Memory[6]		

Reducing Cache Misses by More Flexible Placement of Blocks

Misses and Associativity in Caches

Assume there are three small caches, each consisting of four one-word blocks. One cache is fully associative, a second is two-way set-associative, and the third is direct-mapped. Find the number of misses for each cache organization given the following sequence of block addresses: 0, 8, 0, 6, and 8.

Fully-associative

Address of memory block accessed	Hit or miss	Contents of cache blocks after reference			
		Block 0	Block 1	Block 2	Block 3
0	miss	Memory[0]			
8	miss	Memory[0]	Memory[8]		
0	hit	Memory[0]	Memory[8]		
6	miss	Memory[0]	Memory[8]	Memory[6]	
8	hit	Memory[0]	Memory[8]	Memory[6]	

Reducing Cache Misses by More Flexible Placement of Blocks

Direct mapping – 5 misses
2-way set associative – 4 misses
Fully associative – 3 misses

For this series of references, three misses is the best we can do, because three unique block addresses are accessed.

Notice that if we had eight blocks in the cache, there would be no replacements in the two-way set-associative cache, and it would have the same number of misses as the fully associative cache.

Similarly, if we had 16 blocks, all 3 caches would have the same number of misses.

Even this trivial example shows that cache size and associativity are not independent in determining cache performance.

Locating a Block in the Cache

Consider the task of finding a block in a cache that is set associative.

- As in a direct-mapped cache, each block in a set-associative cache includes an address tag that gives the block address.
- The tag of every cache block within the appropriate set is checked to see if it matches the block address from the processor.

FIGURE 5.17 The three portions of an address in a set-associative or direct-mapped cache. The index is used to select the set, then the tag is used to choose the block by comparison with the blocks in the selected set. The block offset is the address of the desired data within the block.

- Index value - selects the set containing the address of interest, and the tags of all the blocks in the set must be searched. Because speed is of the essence, all the tags in the selected set are searched in parallel.
- As in a fully associative cache, a sequential search would make the hit time of a set-associative cache too slow.

Locating a Block in the Cache

- If the total cache size is kept the same, increasing the associativity increases the number of blocks per set, which is the number of simultaneous compares needed to perform the search in parallel: each increase by a factor of 2 in associativity doubles the number of blocks per set and halves the number of sets.

- Accordingly, each factor-of-2 increase in associativity decreases the size of the index by 1 bit and increases the size of the tag by 1 bit.

- In a fully associative cache, there is effectively only one set, and all the blocks must be checked in parallel. Thus, there is no index, and the entire address, excluding the block offset, is compared against the tag of every block. In other words, we search the entire cache without any indexing.

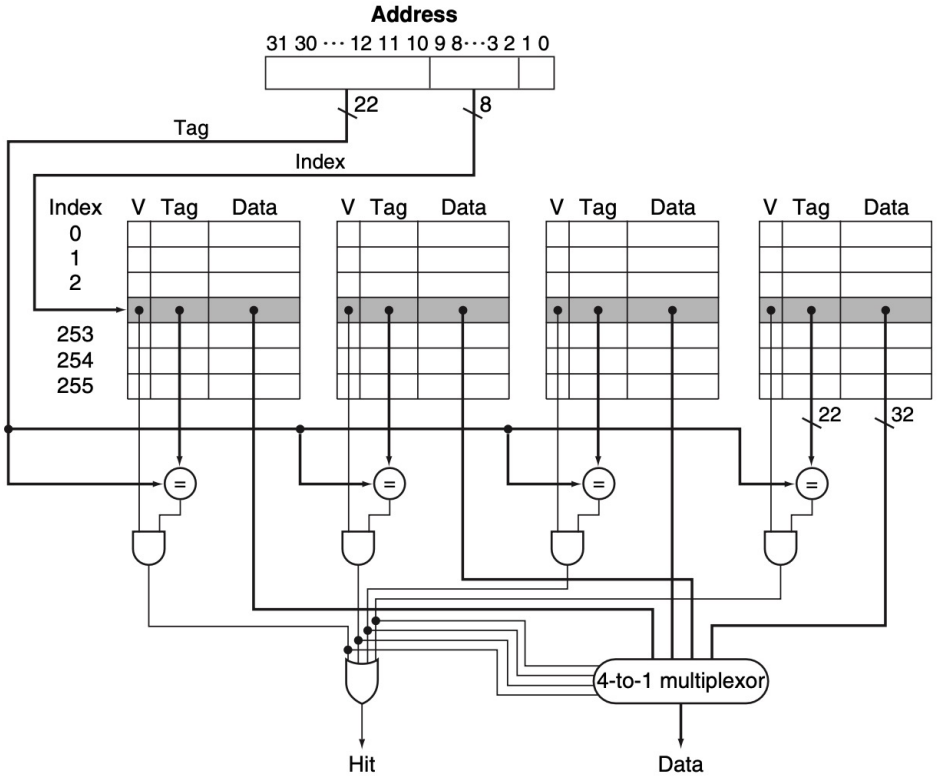
- In a direct-mapped cache, only a single comparator is needed, because the entry can be in only one block, and we access the cache simply by indexing.

Locating a Block in the Cache

Four-way set-associative cache,

- four comparators
- 4-to-1 multiplexor

The cache access consists of indexing the appropriate set and then searching the tags of the set.



- The costs of an associative cache are the extra comparators and any delay imposed by having to do the compare and select from among the elements of the set.
- Choosing the mapping will depend on the cost of a miss versus the cost of implementing associativity, both in time and in extra hardware.

Locating a Block in the Cache

Size of Tags versus Set Associativity

Increasing associativity requires more comparators and more tag bits per cache block. Assuming a cache of 4096 blocks, a 4-word block size, and a 32-bit address, find the total number of sets and the total number of tag bits for caches that are direct mapped, two-way and four-way set associative, and fully associative.

Since there are 16 (2^4) bytes per block, a 32-bit address yields $32-4 = 28$ bits to be used for index and tag.

The direct-mapped cache has the same number of sets as blocks, and hence 12 bits of index, since $\log_2(4096) = 12$; hence, the total number is $(28 - 12) \times 4096 = 16 \times 4096 = 66$ K tag bits.

Locating a Block in the Cache

Size of Tags versus Set Associativity

Each degree of associativity decreases the number of sets by a factor of 2 and thus decreases the number of bits used to index the cache by 1 and increases the number of bits in the tag by 1.

For a two-way set-associative cache, there are 2048 sets, and the total number of tag bits is $(28 - 11) \times 2 \times 2048 = 34 \times 2048 = 70$ Kbits.

For a four-way set-associative cache, the total number of sets is 1024, and the total number is $(28 - 10) \times 4 \times 1024 = 72 \times 1024 = 74$ K tag bits.

For a fully associative cache, there is only one set with 4096 blocks, and the tag is 28 bits, leading to $28 \times 1 \times 4096 = 115$ K tag bits.

Choosing Which Block to Replace

When a miss occurs in a

direct-mapped cache - requested block can go in exactly one position, and the block occupying that position must be replaced.

Set-associative cache - we must choose among the blocks in the selected set.

fully associative cache - all blocks are candidates for replacement.

- Most commonly used scheme is least recently used (LRU).
- In an LRU scheme, the block replaced is the one that has been unused for the longest time.
- LRU replacement is implemented by keeping track of when each element in a set was used relative to the other elements in the set.

Reducing the Miss Penalty Using Multilevel Caches

- All modern computers make use of caches.
- To close the gap further between the fast clock rates of modern processors and the increasingly long time required to access DRAMs, most microprocessors support an additional level of caching.
- This second-level cache is normally on the same chip and is accessed whenever a miss occurs in the primary cache.
- If the second-level cache contains the desired data, the miss penalty for the first-level cache will be essentially the access time of the second-level cache, which will be much less than the access time of main memory.
- If neither the primary nor the secondary cache contains the data, a main memory access is required, and a larger miss penalty is incurred.

Reducing the Miss Penalty Using Multilevel Caches

Performance of Multilevel Caches

Suppose we have a processor with a base CPI of 1.0, assuming all references hit in the primary cache, and a clock rate of 4 GHz. Assume a main memory access time of 100 ns, including all the miss handling. Suppose the miss rate per instruction at the primary cache is 2%. How much faster will the processor be if we add a secondary cache that has a 5 ns access time for either a hit or a miss and is large enough to reduce the miss rate to main memory to 0.5%?

The miss penalty to main memory is

$$\frac{100 \text{ ns}}{0.25 \dfrac{\text{ns}}{\text{clock cycle}}} = 400 \text{ clock cycles}$$

Reducing the Miss Penalty Using Multilevel Caches

Performance of Multilevel Caches

The effective CPI with one level of caching is given by

$$\text{Total CPI} = \text{Base CPI} + \text{Memory-stall cycles per instruction}$$

For the processor with one level of caching,

$$\text{Total CPI} = 1.0 + \text{Memory-stall cycles per instruction}$$

$$= 1.0 + 2\% \times 400 = 9$$

With two levels of caching, a miss in the primary (or first-level) cache can be satisfied either by the secondary cache or by main memory.

The miss penalty for an access to the second-level cache is $\dfrac{5 \text{ ns}}{0.25 \dfrac{\text{ns}}{\text{clock cycle}}} = 20 \text{ clock cycles}$

Reducing the Miss Penalty Using Multilevel Caches

Performance of Multilevel Caches

If the miss is satisfied in the secondary cache, then this is the entire miss penalty.

If the miss needs to go to main memory, then the total miss penalty is the sum of the secondary cache access time and the main memory access time.

Thus, for a two-level cache, total CPI is the sum of the stall cycles from both levels of cache and the base CPI:

$$\text{Total CPI} = 1 + \text{Primary stalls per instruction} + \text{Secondary stalls per instruction}$$

$$= 1 + 2\% \times 20 + 0.5\% \times 400 = 1 + 0.4 + 2 = 3.4$$

Thus, the processor with the secondary cache is faster by

$$\frac{9}{3.4} = 2.6$$

Reducing the Miss Penalty Using Multilevel Caches

- The design considerations for a primary and secondary cache are significantly different.
- In particular, a two-level cache structure allows the primary cache to focus on minimizing hit time to yield a shorter clock cycle or fewer pipeline stages, while allowing the secondary cache to focus on reducing the miss rate and miss penalty.
- The effect of these changes on the two caches can be seen by comparing each cache to the optimal design for a single level of cache.
- In comparison to a single-level cache, the primary cache of a multilevel cache is often smaller.
- The primary cache may use a smaller block size, to go with the smaller cache size and also to reduce the miss penalty. In comparison, the secondary cache will be much larger than in a single-level cache, since the access time of the secondary cache is less critical.
- With a larger total size, the secondary cache may use a larger block size than appropriate with a single-level cache. It often uses higher associativity than the primary cache given the focus of reducing miss rates.