Cache Optimizations

Cache Optimizations (How to Improve Cache Performance?)

- AMAT = Hit time + Miss rate * Miss penalty

- Reducing the miss rate
  - larger block size,
  - larger cache size, and
  - higher associativity.
- Reducing the miss penalty
  - multilevel cache and
  - giving reads priority over write.
- Reducing the hit time
  - avoiding address translation when indexing the cache.

Reduce Miss rate

- A model that sorts all misses into three simple categories: 3Cs
- Compulsory Miss

  The very first access to a block cannot be in the cache, so the block must be brought into the cache. These are also called cold-start misses or first-reference misses. This type of miss occurs even in an infinite cache due to initially the cache is empty.
- Capacity Miss
  - If the cache cannot contain all the blocks needed during the execution of a program, capacity misses will occur because of blocks being discarded and later retrieved. This type of miss occurs even in fully associative cache.
- Conflict Miss
  - If the block placement strategy is set associative or direct mapped, conflict misses will occur because a block may be discarded and later retrieved if too many blocks map to its set. These misses are also called collision misses or interference misses.

First Optimization: Larger block size to reduce miss rate

- Larger blocks take the advantage of spatial locality.
- Larger blocks reduce the compulsory misses (due to spatial locality).
- Larger blocks increase the miss penalty.
- Larger blocks reduce the number of blocks in the cache, it may increase conflict misses and even capacity misses if the cache is small.
- Clearly, there is little reason to increase the block size to such a size that it *increases the miss rate.*
- *There is also* no benefit to reducing miss rate because it increases the average memory access time.
- The increase in miss penalty may outweigh the decrease in miss rate.
- High latency and high bandwidth encourage large block size since the cache gets many more bytes per miss for a small increase in miss penalty.

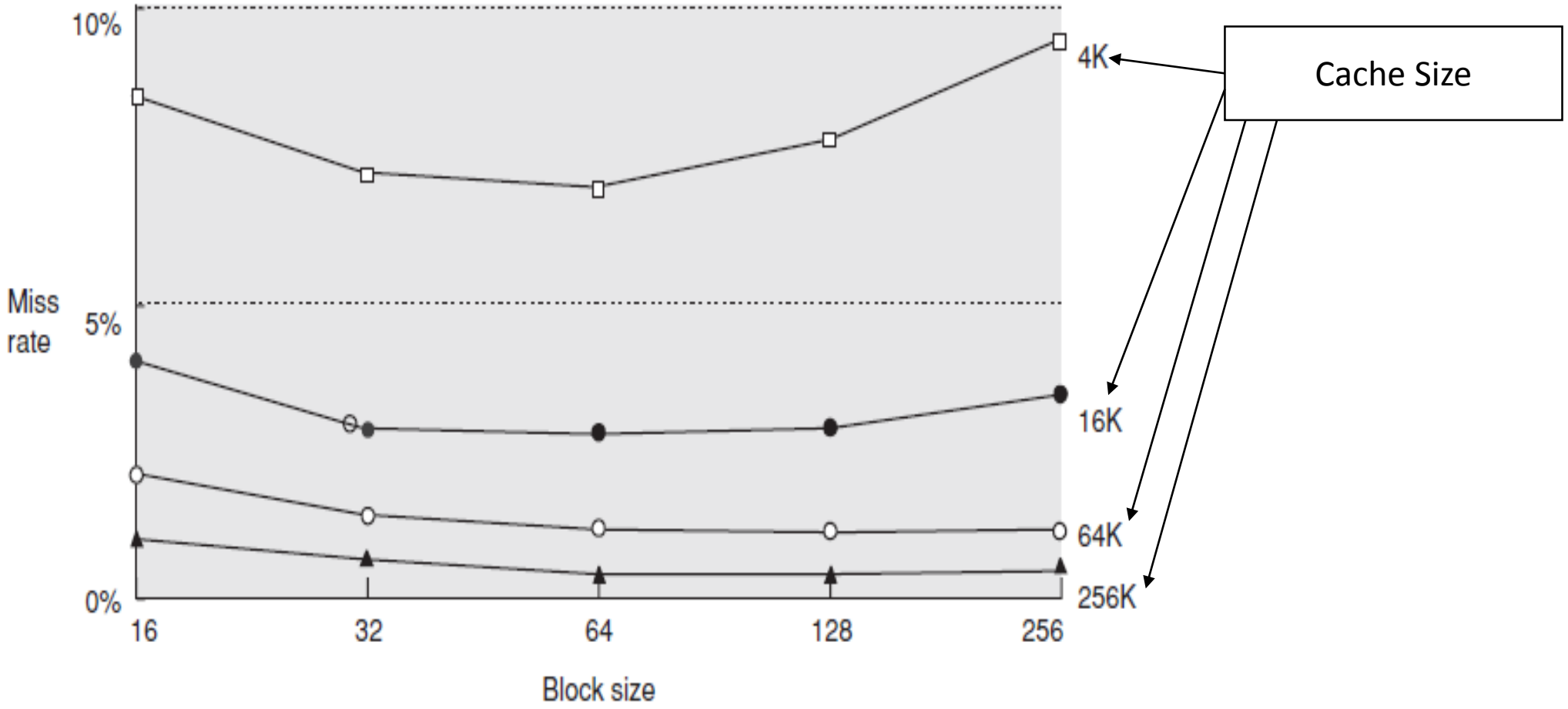

Figure C.10 Miss rate versus block size for five different-sized caches. Note that miss rate actually goes up if the block size is too large relative to the cache size. Each line represents a cache of different size. Figure C.11 shows the data used to plot these lines. Unfortunately, SPEC2000 traces would take too long if block size were included, so these data are based on SPEC92 on a DECstation 5000 [Gee et al. 1993].

Second Optimization: Larger caches to reduce miss rate

- The obvious way to reduce capacity misses is to increase capacity of the cache.
- The obvious drawback is potentially longer hit time and higher cost and power. This technique has been especially popular in off-chip caches.

Third Optimization: Higher associativity to reduce miss rate

- miss rates improve with higher associativity. (reduce the Conflict miss)
- But increase hit time.
- There are two general rules of thumb that can be gleaned from many observations.
- The first is that eight-way set associative as effective in reducing misses as fully associative.
- The second *is that a direct mapped* cache of size *N has about the same miss rate as a two-way set-associative* cache of size *N/2(called 2:1 cache rule).*
- So, improving one aspect of the average memory access time comes at the expense of another.
- Increasing block size reduces miss rate while increasing miss penalty.
- greater associativity can come at the cost of increased hit time.

Example-1

- Assume higher associativity would increase the clock cycle time as listed below:
- Clock cycle time2-way = 1.36 × Clock cycle time1-way
- Clock cycle time4-way = 1.44 × Clock cycle time1-way
- Clock cycle time8-way = 1.52 × Clock cycle time1-way
- hit time is 1 clock cycle, that the miss penalty for the direct mapped cache is 25 clock cycles to a level 2 cache (see next subsection) that
- for which cache sizes are each of these three statements true?
- Average memory access time8-way < Average memory access time4-way
- Average memory access time4-way < Average memory access time2-way
- Average memory access time2-way < Average memory access time1-way

Solution

- ***Answer Average memory access time for each associativity is***
- Average memory access time8-way = Hit time8-way + Miss rate8-way × Miss penalty8-way
- Average memory access time8-way = 1.52 + Miss rate8-way × 25
- Average memory access time4-way = 1.44 + Miss rate4-way × 25
- Average memory access time2-way = 1.36 + Miss rate2-way × 25
- Average memory access time1-way = 1.00 + Miss rate1-way × 25
- If miss rate of 4KB direct mapped cache = 0.098,

  then AMAT 1-way= 1+0.098*25=3.44.
- If miss rate of 512 KB 8-way set associative cache=0.006,

  then AMAT 8-way= 1.52+0.006*25=1.66

Fourth Optimization: Multilevel Caches to reduce miss penalty

- Performance gap between processors and memory.
- Caches should be faster to keep pace with the speed of processors, and cache should be larger to overcome the widening gap between the processor and main memory.
- Add another level of cache between the cache and memory.
- The first level cache (L1) can be small enough to match the clock cycle time of the fast processor. [Low hit time]
- The second level cache (L2) can be large enough to capture many accesses that would go to main memory, thereby lessening the effective miss penalty. [Low miss rate]

$AMAT = Hit\ Time_{L1} + Miss\ Rate_{L1} \times Miss\ Penalty_{L1}$

$Miss\ Penalty_{L1} = Hit\ Time_{L2} + Miss\ Rate_{L2} \times Miss\ Penalty_{L2}$

$AMAT = Hit\ Time_{L1} + Miss\ Rate_{L1} \times (Hit\ Time_{L2} + Miss\ Rate_{L2} \times Miss\ Penalty_{L2})$

- Local miss rate— misses in this cache divided by the total number of memory accesses to this cache (Miss $rate_{L2}$)
- Global miss rate—misses in this cache divided by the total number of memory accesses generated by the CPU
- L1 Global miss rate = L1 Local miss rate
- L2 Global miss rate = Local $Miss\ Rate_{L1} \times Local\ Miss\ Rate_{L2}$

Average Memory stalls per instruction = $Misses\ per\ instruction_{L1} \times Hit\ time_{L2} + Misses\ per\ instruction_{L2} \times Miss\ Penalty_{L2}$

or

Average Memory stalls per instruction =(AMAT - Hit TimeL1)*Avg. memory access per Instruction

Example-1:

- For 1000 memory references :
  - 40 misses in L1,
  - 20 misses in L2
- L1 hit time: 1 cycle,
- L2 hit time: 10 cycles,
- L2 miss penalty=200
- 1.5 memory references per instruction
- Assume ideal CPI=1.0

Find: AMAT, Average stall cycles per instruction ?

- (Local/Global)Miss rate of L1 cache= 40/1000=0.04=4%
- Local Miss rate of L2 cache= 20/40=0.5=50%
- Global Miss rate of L2 cache= 0.04*0.5=0.02=2%
- Average memory access time = $\text{Hit time}_{L1} + \text{Miss rate}_{L1} \times (\text{Hit time}_{L2} + \text{Miss rate}_{L2} \times \text{Miss penalty}_{L2})$

= 1 + 4% × (10 + 50% × 200) = 1 + 4% × 110 = 5.4 clock cycles

- To see how many misses we get per instruction, we divide 1000/1.5 yields 667 instructions. (667 instructions have 1000 memory references)
- Total number of misses per 1000 instruction in L1 and L2 cache=??
- 667 instructions have 40 misses in L1 cache.
- 1000 instructions have 40*1000/667=40*1.5=60 misses per 1000 instruction in L1 cache.
- 667 instructions have 20 misses in L2 cache.
- 1000 instructions have 20*1000/667=20*1.5=30 misses per 1000 instruction in L2 cache.
- Average memory stalls per instruction = Misses per instructionL1 × Hit timeL2 + Misses per instructionL2 × Miss penaltyL2

  = (60/1000) × 10 + (30/1000) × 200

  = 0.060 × 10 + 0.030 × 200 = 6.6 clock cycles

Or:

- average memory stalls per instruction= (AMAT-Cache Hit timeL1)*Avg. memory ref per Instruction=(5.4 – 1.0) × 1.5 = 4.4 × 1.5 = 6.6 clock cycles

Example-2

- For 1000 memory reference :
  - 40 misses in L1,
  - 20 misses in L2
- L1 hit time: 1 cycle,
- L2 hit time: 10 cycles,
- L2 miss penalty=100 cycles
- 1.5 memory references per instruction
- Assume ideal CPI=1.0
- Find Local miss rate, AMAT, Average memory stalls per instruction with and without L2 cache, Compare Performance (by adding L2 cache)

Solution:

Without L2 cache:

Global/Local Miss rate of L1 cache=40/1000=4%

AMAT=hit time +miss rate*miss penalty=1+4%*100=5

Average memory stalls per instruction=(AMAT-hit timeL1)* Average memory access per instruction=(5-1)*1.5=6

CPU time=IC*(Ideal CPI + avg. memory stall cycle per instruction)=IC*(1+6) =IC*7

With L2 cache:

Local Miss rate of L2 cache=20/40=50%

AMAT=hit timeL1 +miss rateL1*(hit timeL2+miss rateL2*miss penaltyL2)
=1+4%*(10+50%*100)=3.4

Average memory stalls per instruction=(AMAT-hit timeL1)* Average memory access per instruction=(3.4-1)*1.5=3.6

CPU time=IC*(Ideal CPI + avg. memory stall cycle per instruction)=IC*(1+3.6) =IC*4.6

Performance improvement with L2 cache = Performance using L2/Performance without using L2= CPU time without L2 / CPU time with L2=(IC*7)/(IC*4.6)=7/4.6=1.52

- Inclusive Cache Vs Exclusive Cache ???

Fifth Optimization: Giving priority to read misses over writes to reduce penalty

- If a read miss has to replace a dirty memory block, the normal sequence is write the dirty block to memory and read the missed block. [In a write back scheme]
- Optimization: copy the dirty block to a buffer, read from memory and then write the block → reduces CPU's waiting time on read miss.

```
SW R3, 512(R0)      ;M[512] ← R3   (cache index 0)
LW R1, 1024(R0)     ;R1 ← M[1024] (cache index 0)
LW R2, 512(R0)      ;R2 ← M[512]   (cache index 0)
```

- In write through caches, write buffers may hold the updated values of a block that encountered a read miss.
- Either wait till write buffer is empty or search in write buffer before going to memory.

Sixth Optimization: Avoiding address translation during indexing of the cache to reduce hit time.

- Software uses virtual address (VA), but memory accessed using physical address (PA).
- VA to PA has to be done before cache look up [MMU, TLB].
- First address translation occurs means VA converts to PA, then cache lookup occurs means PA are divided into tag, index and offset.
- Solution: Start indexing with VA.
- In the meantime do address translation and get PA and tag.
- Indexing by virtual address and tagging by physical address (VIPT cache)
- VIPT: Virtually indexed, physically tagged.