Performance of caches

Performance of caches

- A memory hierarchy can substantially improve performance due to locality and the higher speed of smaller memories.

  Example: Suppose a cache is 10 times faster than main memory and suppose a cache can be used 90% of the time. What is the speedup gained by using the cache ?
  Answer:

$$\text{Speedup} = \frac{1.0}{(1. - 0.9) + 0.9/10} = \sim 5.3$$

- Alternative method to above is to expand CPU execution time equation to account for the # of cycles during which the CPU is stalled waiting for a memory access which is the memory stall cycles.

Performance of caches

- CPU execution time =

(CPU clock cycles + Memory stall cycles) *(clock cycle time)

This assumes CPU clock cycles include the time to handle a cache hit, and the CPU is stalled during a cache miss.

- The number of memory stall cycles depend on both the number of misses and the cost per miss, which is called the miss penalty
- Miss penalty is the additional clock cycles to service the miss

- Memory stall cycles = (# of misses) * (miss penalty)
  = (IC) * (misses/instr) * (miss penalty)
  = (IC) * (memory reference/instr) * (miss rate) * (miss penalty)
- Miss rate is the fraction of cache access that result in a miss i.e. the number of accesses that missed divided by number of accesses. This can be measured with cache simulators that take trace of instruction and data reference, simulate the cache behavior to determine which reference is hit and which is miss. Then report hit and miss totals.
- Memory reference/instr can be done since we know how to measure IC; each instruction requires an instruction access and then we can decide if it requires a data access.
- Advantage of this formulation for memory stall cycles is that the components can be measured easily.

Example : A machine has CPI 2.0 when all memory accesses are hit. Data access, i.e. load and stores, is 40% of total instructions. If miss penalty is 25 clock cycles, and miss rate is 2%, how much faster would the machine be if all instructions were cache hits?

Answer: Machine with all cache hits :

CPU execution time =

(CPU clock cycles + memory stall clock cycles) *(clock cycle time)

= (IC * CPI + 0) * (clock cycle time) = IC * 2* (clock cycle time)

Memory stall cycles (for the machine with real cache) =

IC*(memory reference per instruction) *(miss rate) *(miss penalty)

= IC *(140/100) * 0.02 * 25 = IC * 1.4 *.02 * 25 = IC * 0.7

CPU execution time = (IC * 2 + IC * 0.7) * clock cycle time

Speedup = (2.7 * IC * clock cycle)/(2.0 * IC * clock cycle time)

= 1.35

Recap of last few slides

- Memory stall cycles =
  IC * (memory reference/instr) * miss rate * miss penalty

- Miss rate is the fraction of accesses that are not in the cache

- Miss penalty is the additional clock cycles to service the miss

CPU Performance Units

MIPS = millions of instructions per second

= IC / (execution time in second* 10^6)

= clock rate / (CPI * 10^6)

MIPS is not an accurate measure for computing performance among computers :

- MIPS is dependent on the instruction set, making it difficult to compare MIPS of computers with different instruction set
- MIPS varies between programs on the same computer
- MIPS can vary inversely to performance

- $$\text{MFLOPS} = \frac{\text{\# of floating point operations in a program}}{\text{execution time in seconds} * 10^6}$$

- MFLOPS are dependent on the machine and on the program (same program running on different computers would execute a different # of instructions but the same # of FP operations)
- MFLOPS is also not a consistent and useful measure of performance because
  - Set of FP operations is not consistent across machines e.g. some have divide instructions, some don't
  - MFLOPS rating for a single program cannot be generalized to establish a single performance metric for a computer

- Execution time is the principle measure of performance
- Unlike execution time, it is tempting to characterize a machine with a single MIPS, or MFLOPS rating without naming the program, specifying I/O, or describing the version of OS and compilers

Memory hierarchy design

- Programmers want unlimited amount of memory
- Economic solution is memory hierarchy
- Principle of locality says that data and code are not accessed uniformly
- This principle plus the guideline that smaller hardware is faster led to hierarchy based memory
- All data in a level is found in the level below and so on
- Each level maps addresses from a larger memory to a smaller but faster memory higher in the hierarchy
- Microprocessor performance improved 55% per year since 1987, and 35% until 1986; memory performance improved 7% per year
- Clearly there is a processor-memory performance gap that computer architecture must try to close

Caches

- Caches are the first level of the memory hierarchy encountered once the address leaves the CPU
- Block is the minimum amount of info that can be present (hit) or not (miss) in the cache
- Q1: Where can a block be placed in a cache?
- Q2: Which block should be replaced in a cache miss ?
- Q3: What happens on a write ?

- Q1 :Where can a block be placed in a cache ?

Block is the min unit of info that can be present (hit) or not (miss).

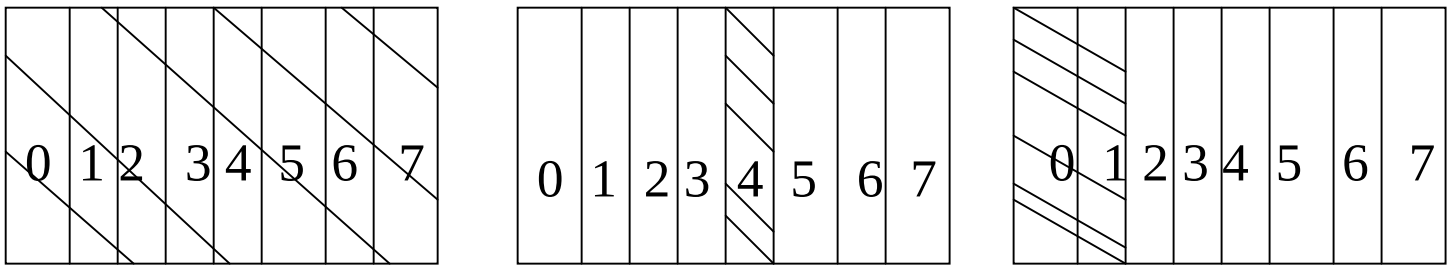
Fully associative:
Block 12 can go anywhere

Direct mapped:
Block 12 can go only into block 4 (12 mod 8)

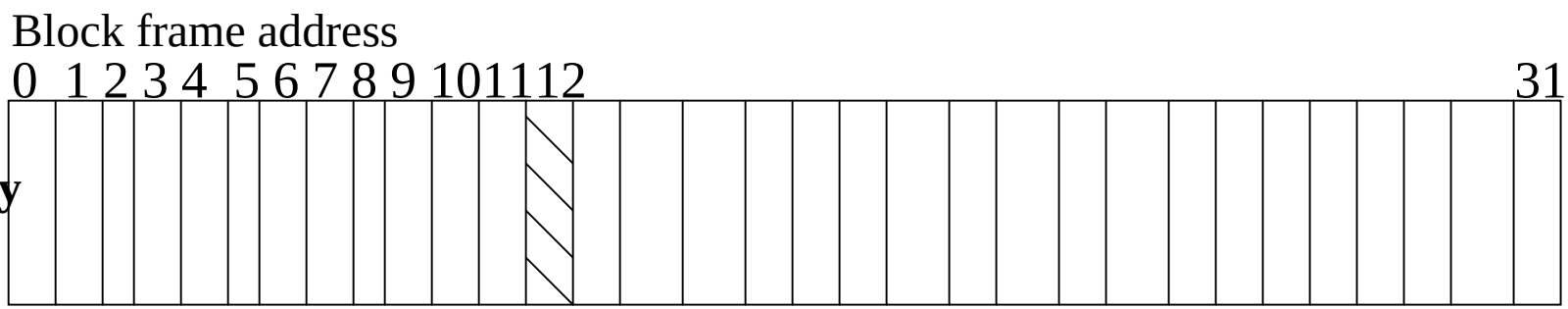
2 way set associative:
Block 12 can go anywhere in set 0 (12 mod 4)

Block #

Cache



Formula : (block address) MOD (# of blocks in cache)



Q2: Which block should be replaced on a cache miss ?

- Direct mapped cache replacement is simpler since only one block frame is checked for a hit and only that block can be replaced
- With fully associative and set-associative there are many to choose from
    - Random : to spread allocation uniformly, candidate blocks are randomly selected
    - Least-recently Used (LRU) : To reduce the chance of throwing out information that will be needed soon, accesses to blocks are recorded. The block replaced is the one that has been unused for the longest
  - Random is simpler to build in hardware. As the # of blocks to keep track increases, LRU becomes increasingly expensive and is frequently only approximated

Q3: What happens on write ?

Reads dominate processor caches. All instruction accesses are reads, and most instructions don't write.

Some benchmarks suggest that in general 9% are stores (writes) and 26% are loads (reads). So writes are (9/(100+26+9)=) ~7% of the overall memory traffic and (9/(26+9)=) ~25% of the data cache traffic

Making the common case fast means optimizing cache reads. Processors wait for read to complete but not writes

Two Write policies :

- Write through (or store through) : The information is written to both the block in the cache and to the block in the lower memory
- Write back (or copy back or store in) : The information is written only to the block in the cache. The modified cache block is written to main memory only when it is replaced

Cache performance examples

- Since instruction count is independent of HW, it is tempting to evaluate performance using that number.
- Correspondingly it is tempting to measure memory hierarchy performance from miss rate since it too is independent of the speed of the HW. Miss rate can be just as misleading as instruction count. A better measure of memory hierarchy performance is the average time to access memory
- Average memory access time = Hit time + (Miss rate * Miss penalty) where hit time is the time to hit the cache. Components of average memory access time can be measured in ns (say for hit time) or # of clock cycles that the CPU waits for the memory such as miss penalty of 50 clock cycles
- Remember the above is still an indirect measure of overall performance, although it is a better measure than miss rate, it is not a substitute for execution time

Example : Miss rates

Size	IC	DC	Unified cache
16kb	.64%	6.47%	
32kb			1.99%

Which has lower miss rate among the two if 75% is instruction and 25% is data references?

Answer : Overall miiss rate for split cache =

(75% * .64% + 25% * 6.47*) = 2.10 %

A 32 kb unified cache has slightly lower i.e. 1.99% miss rate.

Example : Assume 1 hit takes 1 clock cycle, miss penalty is 50 clock cycles; a load or a store hit takes 1 extra clock cycle on a unified cache since there is only one cache port to satisfy two simultaneous requests. What is the average memory access time in each case?

Answer: Average memory access time =

% instr *(hit time + instr miss rate* miss penalty) +
%data *(hit time + data miss rate* miss penalty)
Average memory access time for split cache =
75%(1 + .64%*50) + 25%(1 + 6.47%*50) =
2.05 clock cycles

Average memory access time for unified cache =
75%(1 + 1.99%*50) + 25% (1 + 1 + 1.99%*50) =
2.24 clock cycles

Moral : Miss rate is misleading

CPU time =

(cpu execution clock cycles + memory stall cycles) *clock cycle time

Clock cycles for a cache hit could be part of cpu execution or memory stall cycles. Widely accepted is to include hit clock cycles in cpu execution clock cycles.

CPU time =

IC* ($CPI_{execution}$ +

mem. access per instr. * miss rate * miss penalty)* clock cycle time