Lecture 19
Improving cache performance

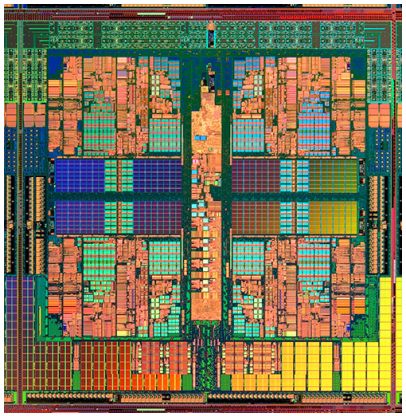
Suggested reading:

(HP Chapter 5.3)

CSE 30321

Processor components

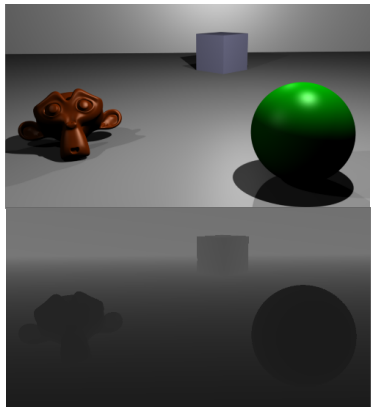
Multicore processors and programming



Processor comparison

AMD Athlon 64 vs. intel Pentium Dual-Core inside

Writing more efficient code



The right HW for the right application

HLL code translation

```
for i=0; i<5; i++ {
        a = (a*b) + c;
}
```

```
MULT r1,r2,r3    # r1 ← r2*r3
ADD r2,r1,r4     # r2 ← r1+r4
```

110011	000001	000010	000011
001110	000010	000001	000100

Fundamental lesson(s)

- How is a physical address mapped to a particular location in a cache?

- How can we improve average memory access time?
  - We'll look at ways to improve hit time, miss rates, and miss penalties

- In a modern microprocessor, there will almost certainly be more than 1 level of cache … and possibly up to 3.
  - We will discuss how data is moved to and from different levels (to try and obtain the best possible program execution times)

Why it's important…

- Memory hierarchies prevent programs from having absolutely abysmal performance
  - In lab 4, you'll see how a detailed understanding of the memory organization can help you write code that is *at least* 2X more efficient

- There is a 100% chance that there will be a question on this topic on your final exam!

- Also, there is a coupling to virtual address translation (to be discussed) which is fundamental in your OS course too.

Cache misses and the architect

- What can we do about the 3 kinds of cache misses?
  - Compulsory, capacity, and conflict…
- Can avoid conflict misses w/fully associative cache
  - But fully associative caches mean expensive HW, possibly slower clock rates, and other bad stuff
- Can avoid capacity misses by making cache bigger – small caches can lead to thrashing
  - With thrashing, data moves between 2 levels of memory hierarchy very frequently – can really slow down performance

Addressing Miss Rates

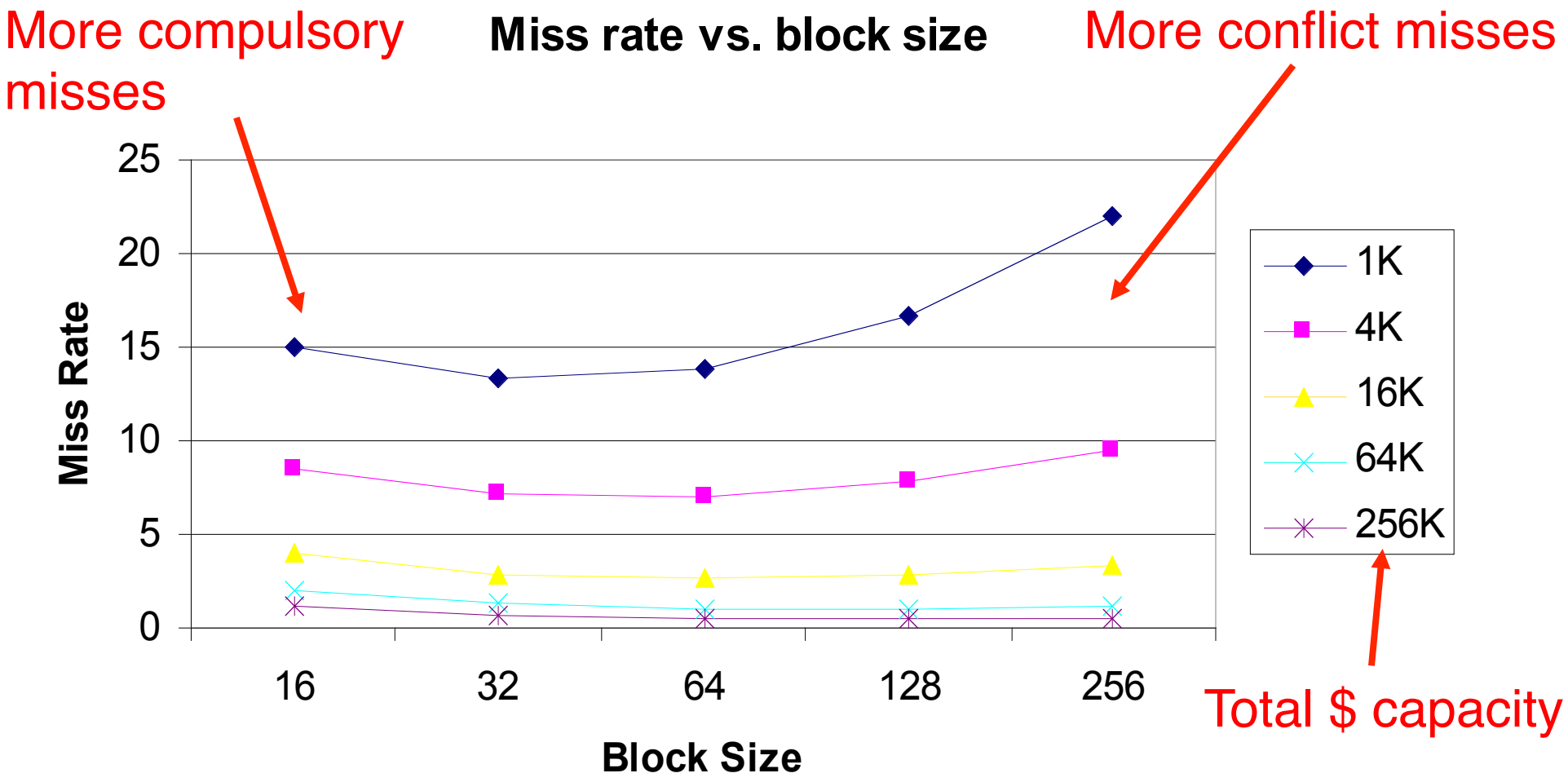
(1) Larger cache block size

- Easiest way to reduce miss rate is to increase cache block size
  - This will help eliminate what kind of misses?

- Helps improve miss rate b/c of principle of locality:
  - Temporal locality says that if something is accessed once, it'll probably be accessed again soon
  - Spatial locality says that if something is accessed, something nearby it will probably be accessed
    - Larger block sizes help with spatial locality

- Be careful though!
  - Larger block sizes can increase miss penalty!
    - Generally, larger blocks reduce # of total blocks in cache

(1) Larger cache block size



(Assuming total cache size stays constant for each curve)

(1) Larger cache block size (example)

- Assume that to access lower-level of memory hierarchy you:
  - Incur a 40 clock cycle overhead
  - Get 16 bytes of data every 2 clock cycles
- i.e. get 16 bytes in 42 clock cycles, 32 in 44, etc…

- Using data below, which block size has minimum average memory access time?

Cache sizes

Miss rates

Block Size	1K	4K	16K	64K	256K
16	15.05%	8.57%	3.94%	2.04%	1.09%
32	13.34%	7.24%	2.87%	1.35%	0.70%
64	13.76%	7.00%	2.64%	1.06%	0.51%
128	16.64%	7.78%	2.77%	1.02%	0.49%
256	22.01%	9.51%	3.29%	1.15%	0.49%

(1) Larger cache block size (cont.)

- Recall that Average memory access time =
  - Hit time + Miss rate X Miss penalty

- Assume a cache hit otherwise takes 1 clock cycle – independent of block size

- So, for a 16-byte block in a 1-KB cache…
  - Average memory access time =
    - 1 + (15.05% X 42) = 7.321 clock cycles
- And for a 256-byte block in a 256-KB cache…
  - Average memory access time =
    - 1 + (0.49% X 72) = 1.353 clock cycles

- Rest of the data is included on next slide…

(1) Larger cache block size (cont.)

Cache sizes

Block Size	Miss Penalty	1K	4K	16K	64K	256K
16	42	7.321	4.599	2.655	1.857	1.485
32	44	6.870	4.186	2.263	1.594	1.308
64	48	7.605	4.360	2.267	1.509	1.245
128	56	10.318	5.357	2.551	1.571	1.274
256	72	16.847	7.847	3.369	1.828	1.353

Red entries are lowest average AMAT for a particular configuration

Note: Data for cache sizes in units of “clock cycles”

(1) Larger cache block sizes (wrap-up)

- We want to minimize cache miss rate & cache miss penalty at same time!

- Selection of block size depends on latency and bandwidth of lower-level memory:

  - High latency, high bandwidth encourage large block size
    - Cache gets many more bytes per miss for a small increase in miss penalty
    - (or … if it takes a long time to process miss, bring in as much information as you can when you do miss)

(1) Larger cache block sizes (wrap-up)

- Low latency, low bandwidth encourage small block size
  - If it doesn't take long to process miss, but bandwidth small, use small block sizes
    - (so miss penalty doesn't go up because of transfer time)
  - Larger # of small blocks may reduce conflict misses

(2) Higher associativity

- Higher associativity can improve cache miss rates…
- Note that an 8-way set associative cache is…
  - …essentially a fully-associative cache
- But, diminishing returns set in sooner or later…
  - Greater associativity can cause increased hit time

Addressing Miss Penalties

Second-level caches

- This technique focuses on cache/main memory interface

- Processor/memory performance gap makes us consider:
  - If we should make caches faster to keep pace with CPUs
  - If we should make caches larger to overcome widening gap between CPU and main memory

- One solution is to do both:
  - Add another level of cache (L2) between the 1st level cache (L1) and main memory
    - Ideally L1 will be fast enough to match the speed of the CPU while L2 will be large enough to reduce the penalty of going to main memory

Second-level caches

- This will of course introduce a new definition for average memory access time:
  - Hit time$_{L1}$ + Miss Rate$_{L1}$ * Miss Penalty$_{L1}$
  - Where, Miss Penalty$_{L1}$ =
    - Hit Time$_{L2}$ + Miss Rate$_{L2}$ * Miss Penalty$_{L2}$
    - So 2nd level miss rate measure from 1st level cache misses…

- A few definitions to avoid confusion:
  - Local miss rate:
    - # of misses in the cache divided by total # of memory accesses to the cache – specifically Miss Rate$_{L2}$
  - Global miss rate:
    - # of misses in the cache divided by total # of memory accesses generated by the CPU – specifically:
      - Miss Rate$_{L1}$ * Miss Rate$_{L2}$

Second-level caches

- **Example:**
  - **In 1000 memory references there are 40 misses in the L1 cache and 20 misses in the L2 cache. What are the various miss rates?**
  - **Miss Rate L1 (local or global): 40/1000 = 4%**
  - **Miss Rate L2 (local): 20/40 = 50%**
  - **Miss Rate L2 (global): 20/1000 = 2%**

- **Local cache rate not good measure of secondary caches – its a function of L1 miss rate**
  - **Which can vary by changing the L1 cache**
  - **Use global cache miss rate to evaluating 2nd level caches!**

Second-level caches

- 2nd level caches are usually BIG!
  - Usually L1 is a subset of L2
  - Should have few capacity misses in L2 cache
    - Only worry about compulsory and conflict for optimizations…

Second-level caches (example)

- Given the following data…
  - 2-way set associativity increases hit time by 10% of a CPU clock cycle
  - Hit time for L2 direct mapped cache is: 10 clock cycles
  - Local miss rate for L2 direct mapped cache is: 25%
  - Local miss rate for L2 2-way set associative cache is: 20%
  - Miss penalty for the L2 cache is: 50 clock cycles

- What is the impact of using a 2-way set associative cache on our miss penalty?

Second-level caches (example)

- Miss penalty / $AMAT_{\text{Direct mapped L2}}$ =
  - 10 + 25% * 50 = 22.5 clock cycles

- Adding the cost of associativity increases the hit cost by only 0.1 clock cycles
  - Thus, Miss penalty / $AMAT_{\text{2-way set associative L2}}$ =
    - 10.1 + 20% * 50 = 20.1 clock cycles

Second-level caches (example)

- However, we can't have a fraction for a number of clock cycles (i.e. 10.1 not possible!)

- We'll either need to round up to 11 or optimize some more to get it down to 10. So…
  - 10 + 20% * 50 = 20.0 clock cycles *or*
  - 11 + 20% * 50 = 21.0 clock cycles
    - (both better than 22.5)

Addressing Hit Time

Reducing the hit time

- Again, recall our average memory access time equation:
  - Hit time + Miss Rate * Miss Penalty
  - We've talked about reducing the Miss Rate and the Miss Penalty – Hit time can also be a big component…

- On many machines cache accesses can affect the clock cycle time – so making this small is a good thing

Small and simple caches

- Why is this good?
  - Generally, smaller hardware is faster – so a small cache should help the hit time…
  - Direct mapping also falls under the category of “simple”
    - Relates to point above as well – you can check tag and read data at the same time!

Example

A cache example:

- We want to compare the following:
  - A 16-KB data cache & a 16-KB instruction cache versus a 32-KB unified cache

Size	Inst. Cache	Data Cache	Unified Cache
16 KB	0.64%	6.47%	2.87%
32 KB	0.15%	4.82%	1.99%

Miss Rates

- Assume a hit takes 1 clock cycle to process
- Miss penalty = 50 clock cycles
- In unified cache, load or store hit takes 1 extra CC
  - (having only 1 cache port = a structural hazard)
- 75% of accesses are instruction references
- What's avg. memory access time in each case?

A cache example continued…

- 1st, let's determine overall miss rate for split caches:
  - (75% x 0.64%) + (25% x 6.47%) = 2.10%
  - This compares to the unified cache miss rate of 1.99%

- We'll use average memory access time formula from a few slides ago but break it up into instruction & data references

- Average memory access time – split cache =
  - 75% x (1 + 0.64% x 50) + 25% x (1 + 6.47% x 50)
  - (75% x 1.32) + (25% x 4.235) = 2.05 cycles

- Average memory access time – unified cache =
  - 75% x (1 + 1.99% x 50) + 25% x (1 + 1 + 1.99% x 50)
  - (75% x 1.995) + (25% x 2.995) = 2.24 cycles

- Despite higher miss rate, access time faster for split cache!