Performance of coherence protocols

Cache misses have traditionally been classified into four categories:

- *Cold misses* (or "compulsory misses") occur the first time that a block is referenced.
- *Conflict misses* are misses that would not occur if the cache were fully associative with LRU replacement.
- *Capacity misses* occur when the cache size is not sufficient to hold data between references.
- *Coherence misses* are misses caused by the coherence protocol.

The first three types occur in uniprocessors. The last is specific to multiprocessors.

To these, Solihin adds *context-switch* (or "system-related") misses, which are related to task switches.

Let's look at a uniprocessor example, a very small cache that has only four lines.

Let's look first at a fully associative cache, because which kind(s) of misses can't it have?

Here's an example of a reference trace of 0, 2, 4, 0, 2, 4, 6, 8, 0.

Fully associative									
	0	2	4	0	2	4	6	8	0
0	0			0				8	
1		2			2				0
2			4			4			
3							6		
	cold	cold	cold	hit	hit	hit	cold	cold	capacity

In a fully associative cache, there are 5 cold misses, because 5 different blocks are referenced.

There are 3 hits.

The remaining reference (the third one to block 0) is not a cold miss.

It must be a capacity miss, because the cache doesn't have room to hold all five blocks.

We'll assume that replacement is LRU; in this case, block 0 replaces the LRU line, which at that point is line 1.

Now let's suppose the cache is 2-way set associative. This means there are two sets, one (set 0) that will hold the even-numbered blocks, and one (set 1) that will hold the odd-numbered blocks.

2-way set-associative									
	0	2	4	0	2	4	6	8	0
0	0		4		2		6		0
1		2		0		4		8	
2									
3									

Since only even-numbered blocks are referenced in this trace, they will all map to set 0.

This time, though, there won't be any hits.

Classify each of these references as a hit or a particular kind of miss.

References that would have been hits in a fully associative cache, but are misses in a less-associative cache, are conflict misses.

Finally, let's look at a direct-mapped cache. Blocks with numbers congruent to 0 mod 4 map to line 0; blocks with numbers congruent to 1 mod 4 map to line 1, etc.

Direct mapped									
	0	2	4	0	2	4	6	8	0
0	0		4	0		4		8	0
1									
2		2			2		6		
3									

Classify each of these references as a hit or a particular kind of miss.

Of the three conflict misses in the set-associative cache, one is a hit here. Block 2 is still in the cache the second time it is referenced. The other two are conflict misses in this cache.

Now, let's talk about coherence misses.

Coherence misses can be divided into those caused by *true sharing* and those caused by *false sharing* (see p. 236 of the Solihin text).

- False-sharing misses are those caused by having a line size larger than one word. Can you explain?

- True-sharing misses, on the other hand, occur when
  - a processor writes into a cache line, invalidating a copy of the same block in another processor's cache,
  - after which

How can we attack each of the four kinds of misses?

- To reduce capacity misses, we can
- To reduce conflict misses, we can
- To reduce cold misses, we can
- To reduce coherence misses, we can

Similarly, context-switch misses can be divided into categories.

- *Replaced* misses are blocks that were replaced while the other process(es) were active.

- *Reordered* misses are blocks that were shoved so far down the LRU stack by the other process(es) that they are replaced soon afterwards (when they otherwise would've stayed in the cache).

Which protocol is best? What cache line size is performs best? What kind of misses predominate?

Simulations

Questions like these can be answered by simulation. Getting the answer right is part art and part science.

Parameters need to be chosen for the simulator. Culler & Singh (1998) selected a single-level 4-way set-associative 1 MB cache with 64-byte lines.

The simulation assumes an idealized memory model, which assumes that references take constant time. Why is this not realistic?

The simulated workload consists of

- six parallel programs (Barnes, LU, Ocean, Radix, Radiosity, Raytrace) from the SPLASH-2 suite and
- one multiprogrammed workload, consisting of mainly serial programs.

Invalidate vs. update

with respect to miss rate

Which is better, an update or an invalidation protocol?

Let's look at real programs.

Where there are many coherence misses,

If there were many capacity misses,

with respect to bus traffic

Compare the

- upgrades in inv. protocol

with the

- updates in upd. protocol

Each of these operations produces bus traffic.

Which are more frequent?

Which protocol causes more bus traffic?

The main problem is that one processor tends to write a block multiple times before another processor reads it.

This causes several bus transactions instead of one, as there would be in an invalidation protocol.

Effect of cache line size

on miss rate

If we increase the line size, what happens to each of the following classes of misses?

- cold misses?
- conflict misses?
- true-sharing misses?

- false-sharing misses?

If we increase the line size, what happens to bus traffic?

So it is not clear which line size will work best.

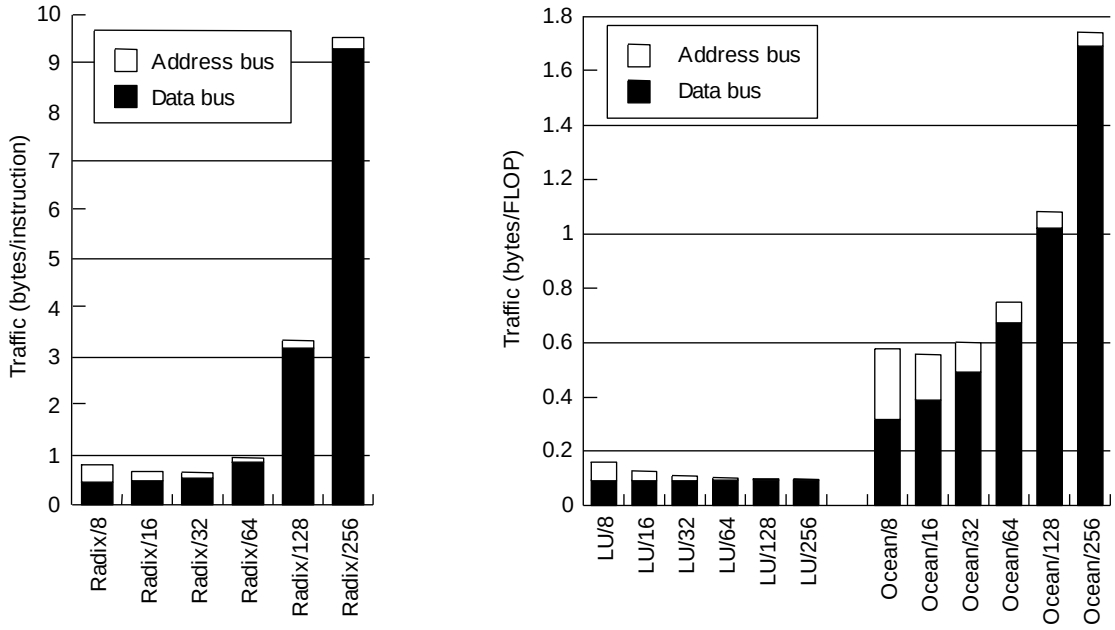
Results for the first three applications seem to show that which line size is best?

For the second set of applications, which do not fit in cache, Radix shows a greatly increasing number of false-sharing misses with

increasing block size.

on bus traffic

Larger line sizes generate more bus traffic.



The results are different than for miss rate—traffic almost always increases with increasing line size.

But address-bus traffic moves in the opposite direction from data-bus traffic.

With this in mind, which line size appears to be best?

Context-switch misses

As cache size gets larger, there are fewer uniprocessor (“natural”) cache misses.

But the number of context-switch misses may go up (mcf, soplex) or down (namd, perlbench).

- Why could it go up?

- Why could it go down?

Reordered misses also decline as the cache becomes large. Why?

Physical cache organization

[Solihin §5.6] A cache is *centralized* (“united”) if its banks are adjacent on the chip.

What are some advantages of a centralized structure?

- Uniform
- Interconnect between the cache and the next level (e.g., on-chip memory controller)

A centralized cache usually uses a crossbar (see also p. 167 of the text).

A cache is *distributed* if its banks are scattered around the chip.

Usually, a portion of the L2 is placed near each L1; this is a *tiled* arrangement.

What are some advantages of a distributed structure?

- In replication:
- In layout:

Hybrid centralized + distributed structure: There's a tradeoff between centralized and distributed.

- A large cache is uniformly slow, especially if it needs to handle coherence.
- A distributed cache requires a lot of interconnections, and routing latency is high if the cache is in too many places.

A compromise is to have an L2 cache that is distributed, but not as distributed as the L1 caches.

Logical cache organization

[Solihin §5.7] Regardless of whether a cache is centralized or distributed, there are several options in mapping addresses to tiles.

- A processor can be limited to accessing a single tile, the one closest to it (private cache configuration).

    - A block in the local cache may also exist in other caches; the copies must be kept coherent by a coherence protocol.
- All of the tiles can form a large logical cache. The address of a block completely determines what tile it is found in (shared 1-tile associative).
    - It may require a lot of hops to get from a processor to the cache.
- A block can be mapped to two tiles (shared 2-tile associative).
    - Block numbers are arranged to improve distance locality.
- Or, a block can be allowed to map to any tile (full tile associativity).
    - What is the upside?
    - What is the downside?

Another option is a partitioned shared cache organization.

- Can you tell how many tiles each block can map to?

- Can you tell how many *lines* each block can map to?

- How does coherence play a role?