CS 61C Spring 2015 Discussion 10 – Cache Coherency

MOESI Cache Coherency

With the MOESI concurrency protocol implemented, accesses to cache accesses appear *serializiable*. This means that the result of the parallel cache accesses appear the same as if there were done in serial from one processor in some ordering.

State	Cache up to date?	Memory up to date?	Others have a copy?	Can respond to other's reads?	Can write without changing state?
Modified	Yes	No	No	Yes, Required	Yes
Owned	Yes	Maybe	Maybe	Yes, Optional	No
Exclusive	Yes	Yes	No	Yes, Optional	No
Shared	Yes	Maybe	Maybe	No	No
Invalid	No	Maybe	Maybe	No	No

1. Consider the following access pattern on a two-processor system with a direct-mapped, write-back cache with one cache block and a two cache block memory. Assume the MOESI protocol is used, with write- back caches, write-allocate, and invalidation of other caches on write (instead of updating the value in the other caches).

Time	After Operation	P1 cache state	P2 cache state	Memory @ 0 up to date?	Memory @ 1 up to date?
0	P1: read block 1	Exclusive (1)	Invalid	YES	YES
1	P2: read block 1	Owned (1)	Shared (1)	Yes	Yes
2	P1: write block 1	Mod (1)	Inv(1)	Yes	No
3	P2: write block 1	Inv(1)	Mod(1)	Yes	No
4	P1: read block 0	Ex(0)	Inv.	Yes	No
5	P2: read block 0	Owned(0)	Shared(0)	Yes	No Yes
6	P1: write block 0	Mod(0)	Inv.	No	Yes
7	P2: read block 0	Own(0)	Shared(0)	No	Yes
8	P2: write block 0	Inv.	Mod(0)	No	Yes
9	P1: read block 0	Shared(0)	Owned(0)	No	Yes

Yes

Concurrency

2. Consider the following function:

```
void transferFunds(struct account *from,
                   struct account *to,
                   long cents)
 {
   from->cents -= cents;
   to->cents += cents;
 }
```

a. What are some data races that could occur if this function is called simultaneously from two (or more) threads on the same accounts? (Hint: if the problem isn't obvious, translate the function into MIPS first)

If the accounts where called simulataneously but with from and to reversed for each call, one possible race could completely overwrite the results of one of the calls.

b. How could you fix or avoid these races? Can you do this without hardware support?

Adding locks to the structs cent variables. Can't do this without hardware support.

If you make the accounts array in a for loop... that forces the access of a accounts sequentially? ... I am still not convinced...

Midterm Questions:

3. Summer '12, MT1, Q1f

In our 32-bit single-precision floating point representation, we decide to convert one significand bit to an exponent bit. How many **denormalized numbers** do we have relative to before? (Circle one)

More [✓ Fewer]

Rounded to the nearest power of 2, how many denorm numbers are there in our new format? (Answer in IEC format)

2^22 whoops, 2^23, since i forgot about sign

4. Fall '14, Final, M2a-d

Assume we are working in a 32-bit virtual and physical address space, byte-address memory. We have two caches: **cache A** is a direct-mapped cache, while **cache B** is fully associative with LRU replacement policy. Both are 4 KiB caches with 256 B blocks and write-back policy. Show all work!

a) For **cache B**, calculate the number of bits used for the Tag, Index, and Offset: T: 24 I: 0 O: 8

Consider the following code:

```
uint32_t H[32768];          // 32768 = 2^15. H is block-aligned.

for (uint32_t i = 0; i < 32768; i += 2048) H[i] += 1;
for (uint32_t i = 1; i < 32768; i += 2048) H[i] += 2;
```

b) If the code were run on **cache A**, what would the hit rate be? read 0, write 100 %

hmmm the answer key says write 50...
i don't agree...

c) If the code were run on **cache B**, what would the hit rate be? read 50, write 100 %

hmmm the answer key says write is 75% for second loop.....

d) Consider several modifications, each to the original **cache A**. How much will the modifications change the hit-rate and why?

i. Same cache size, same block size, 2-way associativity
No change since the associativity get rewritten every other stride

ii. Double the cache size, same block size
No change since it still loops every stride

iii. Same cache size, block size is reduced to 8B

No change since it still loop every stride