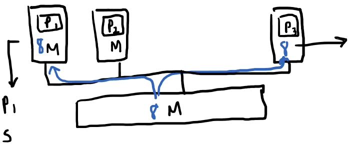
Cache: Snooping & Directory Based

Tuesday, April 5, 2022 7:24 AM

In many processors, we solve the redundance by applying write-through and write-back, the same as with one processor.

The Update of cache memory is done using:

Snooping: used in shared memory with a bus
Directory-based: distributed memory
Example:



commonly, values in cache are called shared

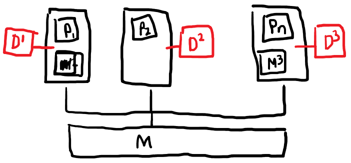
if P_1 wants to change it,

Snooping: sends a msg through the bus to all p's that have the value. Then all p's change their state to invalid. Also RAM
P_1 then changed the value, with Modified state. If another processor wants it, it asks through the bus to update the value (everyone sees it). The modified state processor then updates RAM memory, and P_2 also updates from RAM's new value.

the problem is the bus: if $p \to \infty$, the bus is filled and the waiting time appears due to the bus

In the case of Distributed Mem, the communication goes from device to device: RAM and cache communication is faster and thus no longer a bottleneck
Directory-based: every P has its own directory, used for the cache state.

Let's say P_2 has x, and P_1 wants it, it sends a message. the x is then sent to P_1's cache. Then, P_2 writes on its directory:
x is shared with P_1
If another P_n wants x, it travels to P_n's cache. P_2's directory then updates the directory: x is shared to P_1, P_n



directory: a separate data structure

Imagine P_1 wants to modify x: Sends a message through the net to P_2, so that P_2 invalidates all copies on its own RAM and (through a msg) invalidates all other p's in the directory. Once a p invalidates a value, it needs to send a msg that it can now proceed.
In p_2's directory; x is shared P_1, P_n → invalid. Thus, if another p wants it, P_2 sends a msg to P_1 (correct value) so that P_1 sends the correct value, P_2 corrects it on its RAM and the directory: x in P that asked for it.

it had in its directory: x in P1 (D_i)

The other problem is: False Sharing
↳ this does not happen in 1 processor
It happens when we have a cache per processor

RAM (Shared   cache P_1 cache P_2

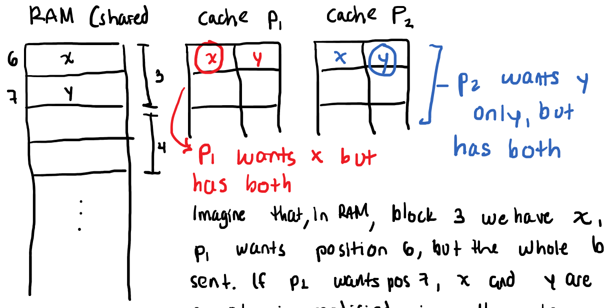
→ how do we avoid False Sharing?

Separate the variables:

instead of modifying x:

```
[ int tempx = x;
  (modifying)
  x = tempx;
```

↳ avoid False Sharing



Imagine that, In RAM, block 3 we have x, and thus p_1 wants position 6, but the whole block is sent. If p_2 wants pos 7, x and y are sent. When a value is modified, since the cache works with lines, it Invalids THE WHOLE LINE. Thus, in the RAM, it needs to invalid the block too. If another processor wants y, therefore p_2 updates RAM value, but with the whole line, and thus needs to also update x from another $\underbrace{\text{processor}}_{P_1}$ = Performance becomes slow.

C++ Multi thread

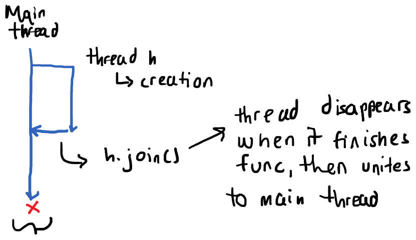
Two ways of creating a thread:

→ Union:

thread h = thread (func, x);

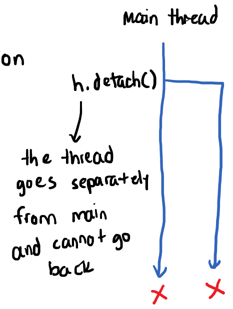
not pointer, the name

params


Main thread
thread h
↳ creation
↳ h.join()
thread disappears when it finishes func, then unites to main thread

If we break the main thread the threads continue.

→ Separation



If we break the main thread, all dettached threads go away too.

$1 << 30 \to 2^{30}$

volatile S s; → volatile since we do not want the compiler to optimize it (instead of the for loop, it just makes s = N), but we want it to execute the for.