Parallel Computer Architecture
Prof. Hemangee K. Kapoor
Department of Computer Science and Engineering
Indian Institute of Technology Guwahati
Week - 04
Lecture - 23

Lec 23: Types of Coherence Protocols (2)

Hello everyone. We are doing module 3 on cache coherence. We are continuing with the lecture on types of cache coherence protocols. In the context of snooping protocol, we are going to see how everything would work and see a big picture of an implementation of snooping protocol before we dive into smaller detailed protocol versions. Okay. So, we have, the setup is that we have multiple processors, multiple caches having probably caching the shared data items and whenever data item is cached by a processor, it could be dirty that is it has been modified or it could be only brought in for reading.

So, it is a read only copy and probably certain caches even do not have it. So, every cache block has a state associated with it within its local cache. The scenario when a processor wishes to write to a shared data item, the other processors having this particular block, what happens to their copies, right? So, those copies will become inconsistent if this processor starts changing it. So, the protocol we are going to follow is this particular cache is going to invalidate all the other sharers copies before changing its own data item.

So, we have to send invalidation request over the broadcast medium, in this case it is the bus. Given a scenario that two caches want to modify the same data item probably parallelly. Now, how will we handle this? Now, both of these will first need to go onto the bus, arbitrate for the bus to send the invalidation request to the other sharers. So, exactly one of them will succeed. So, the one which succeeds invalidates the other copies, makes changes and completes its operation.

So, the second processor will eventually try to retry its request, send a request on the bus and the first processor will invalidate its copy and the second will get a chance to update the data item. So, this way the broadcast medium that is the bus in this case helps to serialize the writes that is the writes will happen one after the other. Okay. Eventually, there will be a new processor wanting to read the data item. Now, which of the caches among this complete system has the most recent data item. So, we want to find out that when a new reader comes into the system, where is the latest copy of the data.

So, to see how the whole system works, first thing we need a cache and every cache maintains state of the cache block valid, invalid, modified, dirty all those different states. Whenever it wants to change the data, it has to send an invalidation request over the broadcast medium so that other cached copies get removed and this particular cache only writes to the data. If two processors want to write, then they have to arbitrate to the bus and they will be able to write one after the other that is we are able to achieve write serialization because of the bus. And whenever a new cache miss occurs for a future data item access, then our question to answer is where do I find the latest copy of the data. Okay. So, we are going to discuss two types of protocols, one is write invalidate and the second is write update.

Write invalidate based cache coherence protocols. As the word write invalidate says that whenever a cache wants to modify the data item, it should invalidate all the other copies throughout the system that is the other sharers of the data item should remove this copy. So that this particular cache has got an exclusive access to this data item. Right. So, with this exactly one processor will be able to change the data item. What is the negative side of this particular design decision is that if there are other readers into the system, future reads to this block will incur a cache miss because this particular write has removed those data items.

So, the new reader will incur a cache miss and will again have to go out to read the block again. But the positive aspect of this particular protocol is that once a cache invalidates other copies, it has exclusive permission and it can continue to write on this. So, multiple writes can be done locally without going on the interconnect. So, we save on the interconnect bandwidth and no additional traffic gets generated. Another positive point is if there are some caches having this block, but they are not using it.

So, such useless copies of data items will be automatically removed by this invalidation message. Right. So, invalidation protocol essentially says that exactly one cache will have the data item for writing, the other copies are removed. It has small disadvantage that future readers will incur a miss. But the positive point is that multiple writes can be done in one go. So, no additional traffic occurs on the interconnect and any so called dead cache blocks will get cleared out automatically because of this.

So, we will quickly see an example. So, you have two processors A and B and they have their respective caches, variable X sitting in the memory with value 0. Now, A wants to read X. So, A wants to read X. So, it will come to the memory and then go back with the value of X equal to 2.

Later B wants to read X. So, the third step is B reads X equal to 0 in the third step. Now, A wishes to write X equal to 1. So, this one says make X equal to 1. Now, when we want to do this although this is a cache hit, but we do not have permission to write this data because there are probably other sharers in the system.

So, when this happens A has to inform everybody on the network that I am going to change X and therefore, B's copy will be removed. So, B has to remove its copy of X before allowing A to write. And later when B again wants to read it will incur a cache miss. So, it will incur a miss and it has to go out again on the bus to fetch the data item. So, the write invalidation protocol whenever a new cache wants to write the data that has changed the content it has to invalidate all the other copies throughout the system.

So, in the context of invalidation protocol we have the concept of an exclusive state because mostly with invalidation protocols we are going to use a write back cache as we have the block with us we can keep on changing the data and eventually write it back to the main memory. So, normally the cache will be a write back cache. In a uniprocessor if we want to write to a block we go to a memory, bring the block to the cache, keep changing it several times and eventually remove it from the cache and write to the main memory. We do not have to inform anybody else, but in a multiprocessor cache coherent system whenever up cache modifies the data item it has to inform others telling them that you do not access it. I am going to change it. So, acquiring this permission is like acquiring ownership.

So, this particular cache tries to become owner of that particular data item and this ownership is similar to having an exclusive access. So, when I have exclusive access to something I do not have to inform others and I can continue to update the data item without telling others. Okay. So, this concept of exclusivity is coming into picture when I am having an invalidation based protocol. So, the particular cache acts as an owner of that data block and future reads to this block who is going to give the data item, this particular owner. So, supplying the data item becomes the responsibility of this particular exclusive copy of the data block.

So, that is the concept of exclusivity. Now consider a scenario where the cache has the block, but it has brought it initially to do a reading. So, when you bring the block to read we have not informed others to invalidate because everybody can read at the same time, but in future if the same processor wishes to write to the block, then we have a new type of a transition which is called a write miss request. Okay. So, assume the scenario that block is present in the cache, so block B, but this was brought only for reading. So, we had brought this block for reading and now we want to change this block, that is update data inside this.

So, we want to write, but before writing we have to guarantee that others have removed their copies and hence we have to do some extra work because this type of request is called a write miss request. Because actually it is a cache hit because the block is there, but it is hitting on a read whereas we wish to write and hence, for writing I do not have permission right now and hence it is called a write miss. So, write miss in an invalidation protocol, informs others of the impending write. So, when a write miss occurs, this particular processor sends a write miss request on the interconnect to tell others that I am going to change this and what is the impact of this write miss? Every other cache will remove its copies that is they will invalidate the copy, this particular cache will get exclusive access to this block which it already had, but it had it for reading. Now it wants to write and hence a write miss transaction has to be handled.

Now in this case again if multiple processors want to write then they will have to do this write miss transaction one by one on the bus which will then serialize it. Subsequent accesses to this block who is going to give you the latest data copy? Either the one which has changed or in case eventually this particular cache removes this block because of cache eviction this data will be updated in the main memory and future readers will take it from the main memory. So, that is write invalidation where we remove all the copies. Now the other variation is write update protocol. What does this mean that whenever a cache writes to a data item it does not remove other copies.

It says that when I change the data item I am going to broadcast the new change across the medium so that other copies will again update themselves. So, if I am making X from 0 to 5, I will broadcast on the bus saying that X has become 5 and because of this all the other caches which have this value X stored inside them, they will pick up this new value of 5 and update their caches. So, that is the meaning of a write update protocol. So, as we can understand that this is going to take considerable bandwidth because every small write is going on to the bus compared to the invalidation based protocols, lot of bandwidth is required. But the positive aspect of this is that future readers that the subsequent readers will not incur a cache miss because they will always find the block inside their cache because they keep on updating that block with the newer values.

But the negative aspect is that multiple useless updates go because if I am changing X to 2 then to 5 then to 6 then to 7. So, I am doing several writes and all of these writes are going on to the bus whereas actually only the last write could have been useful to others. Right. So, we have several useless writes being broadcasted in a write update protocol compared to write invalidate protocol. Okay. So, we will again redo the example with the write update. So, here what happens X equal to 0 comes here, X equal to 0 comes here in the first two steps.

Now when X is made 1, so A makes X equal to 1 it does not remove that copy but rather it sends X equal to 1 as a message here. So, that this one updates its value to 1 and similarly it is also possible depends on the protocol that the memory can also pick up the new value. So, with the write update protocol all the sharers as well as the memory will have the most recent value. Okay. So, this is how a write update protocol will maintain coherence. Okay. The last question was where do I find the latest copy of the block? That is for future readers when we have done all this write update, write invalidate, coherence is maintained but a new reader comes into the system where does that reader find the data block, where is the most recent value available.

Now this depends on the type of cache with every processor that is either the cache is write through or it is write back and then we have two types of protocols write invalidate and write update. So, we need to look at all these options. Okay. So, we will start with write through. So, first the cache was write through and write through caches are very easy you only update and then changes always go to the next level. So, the main memory is always up to date.

Negative aspect is that it uses lot of bandwidth and can increase the latency of the local processor. But if I have a write through cache and I am using a write update protocol. Now look at this combination write through cache, write update protocol. Update meaning you have to update every other copy in the system and anyway the cache is write through. So, when you change anything your write is going through and through to the main memory as well as it is also travelling on the bus.

So, the same transaction can be picked up by the other sharers. So, in a write through write update protocol what happens? Write through write update what will happen here? The memory is up to date and when you are updating the memory, the sharers would also have picked up the data item. So, when I am sending X equal to 1 to the memory, right? in a write through cache. This is a write through cache it is only interested in sending on the memory. But because I have a write update protocol all the other caches will also pick up this new value of X equal to 1.

So, essentially everybody in the system has got the most up to date copy and so how do I answer this question where is the latest copy of the block? It is with all the sharers as well as the memory. Okay. Next combination is write through with write invalidate protocol. So, I have a write through cache which sends the data to the next level of memory, but it is an invalidation based protocol. So, here the other sharers are not going to pick up the value because they do not have the block. Only the writer and the memory has the data block and hence the latest copy will be available with the owner that is the

one who has the exclusive permission of the block and the main memory. Right.

So, now we will see the other case that I have a write back cache and then these two types of protocols. So, when I have a write back cache all the local updates or local writes will happen in this particular block and only when this block is removed from the cache will it be updating the memory. Okay. Now in a write back with a write update protocol any write back cache will keep on changing the data in its local cache and being a write update protocol all the values travel on the bus. Okay. So, let us see this scenario write back with write update. The same thing as we saw in the previous slide, this is a cache, but here all the values keep on collecting in this.

So, whatever changes we do get collected in the cache, but they also get propagated to other processors. So, other processors will also keep getting the newer values, but here the memory does not yet unless the memory picks up the value from the bus the memory is not supposed to know this value. When will the memory get the up to date value? Only when this block is removed from the cache. Right. So, when this one writes back the cache only then x equal to 1 will go to the memory until then the memory may or may not have the up to date value of the data item. Okay. So, write update protocol who has the correct value? Definitely all the sharers and in case the protocol make sure that the memory also can pick up the data when the write update is happening the memory will also have the up to date data item.

So, the latest copy is with all the sharers and probably also with the memory. What happens in the write invalidate protocol? Write invalidate local copy gets changed more often other sharers do not have the data, it is a write back cache. So, even the memory does not have the data. So, all the updates are happening here, the other caches do not have the data item and the memory also does not have the up to date copy. So, memory also has the old data value and all the new values of X are sitting here 1 then it becomes 2 then it becomes 3 all the new values are here.

So, when this is the scenario and a new block wants to read x where will this processor find the value of X? If it goes to the memory no memory has the old data. So, it has to find the owner in the system to identify the correct value. So, this is a slow retrieval because the new reader has to go into the cache of another processor to acquire the latest value of the data. So, latest copy in a write back write invalidate protocol is in some other cache or actually the cache of the owner only. Okay. So, we are discussing about every cache being able to snoop.

Now let us quickly see how does this actually snooping happens and does it affect the behavior of the local system in which the snooping is implemented. Okay. So, this

picture is showing us that there is a bus and there is a bus snooper, that bus snooper talks with the cache and then the processor also talks with the cache. Any transaction on the bus has to be snooped by the snooper and what does snooping actually mean, that whatever address is going on the bus it has to compare whether that address is sitting in my cache also. If this particular cache has that address then we need to either copy that data in case of write update or we need to invalidate our copy in case of write invalidate protocol. So, an action has to be taken if that address matches.

Now for address matching the tag or the address is going on the bus and the snooper also has to check that with the cache. Right? So, an address is coming here. Now this address the snooper has to compare. It has to compare with the address stored in the cache.

There is an address kept here. So, all this has to be compared by the snooper. Now when the snooper is accessing the tag array in case the processor also wants to read the data inside the cache, it also has to do a tag compare. So, now there is a contention that the same cache, the same tag array is being accessed by both the processor and the snooper, right. So, processor also wants to do tag compare and snooper also wants to do tag compare. So, this is going to create contention and one of them will stall and in any case we do not want to stall the processor.

So, how do I solve this problem? I want to snoop definitely, but at the same time the processor also should not stop. So, one solution to this is if I can duplicate the tag array because this tag storage is very tiny, if I can have just a copy of this, right. So, just maintain two copies of the tag arrays for the cache and give one copy to the processor to compare and another copy for the snoop. So, this way I can solve the contention issue.

So, there would not be a stall. So, that is solution number one. The other solution is, if I can use a multi level cache. So, instead of duplicating tags which we discussed in the previous slide here I am going to use two levels of cache. So, when I have two levels of cache, mostly the processor is going to only do the tag comparison with respect to the first level and the snooper is going to do the tag comparison with respect to L2. So, there is less competition between the two, that is the snooper only checks the tags of L2 and processor mostly is going to check tags for L1.

The only problems will come whenever I have a miss in L1. So, if there is a L1 cache miss then this miss will have to go and handled by the L2. So, that is the point when the L2 tags will have a contention that is L1 is also accessing the tag array and the snooper is also accessing the tag array. However, we assume that these scenarios will be less frequent and hence I can definitely go for such a two level hierarchy and give one set of

tags for the processor and another set of tags for the snooper. The assumption here is that if you find the tag in L2, then probably that particular block is also there in L1. Right. So, we are assuming an inclusive property here that is L1 is always a subset of L2.

So, if L2 says the tag is there it has to do extra work because if it has to invalidate its copy you understand that it cannot invalidate a copy only in L2 because this particular block may also be cached in L1. So, in case of hits in case snooper finds the tag in L2. So, L2 also has to go to L1 for invalidating that particular copy. So, there is more work in this method however, it is helping us to keep the processor going on. In this example, if there is a tag match and this cache is supposed to provide the data that is it is the owner and it has to give the data on to the bus, but L2 will have the stale copy because this particular block is with L1 which has the most recent copy.

So, in this case L2 will also have to go to L1 to retrieve the data item and then send it on the bus. So, there is slightly more work to be done when the L1 L2 interaction has to take place, but otherwise it is quite fast and good performing option. Okay. So, we have seen many things. So, what do I choose? Right? So, there is no one solution we will just list what are the options. So, what options do I have? I have a snooping protocol or I have a directory protocol.

Then I have an invalidate based protocol and update based protocol. And then different caching strategies are there depending on the type of caching strategy, the state of the cache blocks will be different and the states of the cache block will decide the transition diagram because the finite state machine which we were drawing in the previous lecture all the states will be different depending on the type of cache you are doing. Then the states will also decide the type of actions to do. Right. So, design space is quite large and the definitely the computer architect has got lots of flexibility to do this.

However, there is no one good solution. So, we are going to look at few protocol implementations to see how we can do best with all these options available. Okay. So, for choosing we can choose between these two types of protocols, these variations of them. So, we have four choices and then the caching strategy also has to be decided. So, we have many options and what we are going to see next is example protocols to illustrate various design options. Okay. So, with this we have seen an overview of the types of cache coherence protocols and from next lecture we are going to see different examples.

So, whenever we design a cache coherence protocol, how do we say that that protocol is good. So, end of every design we are going to check whether it satisfies these two properties which are listed here. One is write propagation, that is every write which is

done by one cache, is it visible to everybody els? So, write propagation has to be guaranteed and the second is write serialization that is if two writes happen to the same location and they happen in the order w1 followed by w2 then everybody should see the writes in the same order. So, write propagation and write serialization are the two properties which must be satisfied by every cache coherence protocol. Right.

So, whenever we design we are going to check that this is happening. Okay. So, with this we finish all the types of protocols. Thank you so much.