Parallel Computer Architecture
Prof. Hemangee K. Kapoor
Department of Computer Science and Engineering
Indian Institute of Technology Guwahati
Week - 04
Lecture - 22

Lec 22: Types of Coherence Protocols (1)

Hello everyone. We are doing the module 3 on cache coherence. We have discussed the importance of coherence and the need of it. In this lecture, we are going to understand what are the different types of cache coherence protocols that can be designed to solve that problem. So the key implementation aspect of coherence is that we need to somehow know that others are sharing a cache block. That is there is a shared variable x in the main memory. Because of caching, this variable has gone to several processors and we need to somehow keep track of who is reading or writing to this location.

So keeping track of the state of the shared data block is the important implementation aspect. And based on this, we categorize the coherence protocols into two types. One is snooping and one is directory based. Okay. So snooping cache coherence protocols, as the word snoop says that we go and snoop on something or check what is happening outside. Okay.

So here the sharing state of that particular block is identified by every processor on its own. That is every processor indirectly the cache controller is going to look outside from its domain to see what is happening. So how do I look outside? Essentially on the global interconnect or a bus. So if you imagine there is a bus connected to the processor, so the cache controller is going to keep seeing what is happening on the bus. And whenever a write or a read happens to a shared variable that happens to pass through this bus.

So we are going to see what is happening on this bus and and derive the state of this particular shared variable. Okay. So here the state of every shared variable is known to the individual caches. It is not kept anywhere in the global centralized location. Every cache knows whether the cache has the block, it has changed it, it does not have it or whatever. So every cache knows the state of the block and there is no central position where this information is kept. Okay.

And how do we communicate this individual information to others? By using a broadcast medium. That is if there are three processors sharing a variable X, whenever

they do any change on X they kind of broadcast it. They shout it out that this is what I am going to do at X and everybody listens to this. Okay. So the information about the shared data is kept distributed across all these caches and not in a centralized position. So that is the method used in snooping protocol.

The other method is when I use a directory based protocol. This as the word directory says there is a centralized location in which information about all the shared caches or all the caches which are sharing this variable is kept. Okay. So directory says that there is a single location in which information about the shared variable is maintained. Okay. So snooping versus directory, snooping is the sharing state is kept in every cached copy. Okay. And there is no centralized storage. There is no centralized place of keeping it.

We need a broadcasting medium to inform others about these shared states and on this broadcast medium the cache controller are going to snoop and find out this information. In the directory based protocol the sharing state is kept in a separate location or a centralized location. So this separate location could be in the main memory or it could be kept as a separate module. So what is the advantage of keeping it outside? The advantage is that I can use this for scalability. Because if I am using a snooping based only up to some level of machines because it depends on broadcasting. So I should be able to broadcast information and there is a limit of how much I can go for broadcasting. Because if your processor scales too much, you have hundreds of processors then broadcasting might not be practical and hence then in that case directory becomes a useful option.

So if you want to scale multiple processors you should go for a directory based protocol and not a snooping based protocol. So that is the major difference about snooping versus directory. Snooping stores the states with the local caches, directory stores the state in a centralized directory or storage. Okay. A quick recap of what is coherence? We have already seen this. There is a shared memory location u and several processes are accessing it.

So P1 reads this, P3 also reads, later P3 modifies and then P1 and P2 try to read u. So what is the value they are going to get? Okay. So we will continue this example and say that P1 and P2, right? P1 and P2 do a read on u and before this read happens P3 has changed the u. So how will P1 and P2 know the new value of u? So that is the coherence problem we are going to solve. And in all the discussion which will happen in this lecture, we are not going to discuss from the perspective of a directory based protocol. We are first going to understand from a snooping protocol implementation and once we understand that we look at different snooping protocols and then tackle the directory data structure. Okay.

So right now every discussion will be in terms of snooping that is broadcasting on this common bus. Okay. So P3 has changed your u from 5 to 7 and our job is that how will P1 and P2 know about this? Okay. So state of that block is with every processor. So I will use the elegant nature of the bus. So bus is there for broadcasting. So P3 is going to send that I am changing u somehow on this bus and P1 and P2 are going to observe this message going on to the bus and take appropriate actions.

So we want to ensure that P1 and P2 will see the new value of P3. So P3 wrote u equal to 7 and I want this to be seen by P1 and P2. So I am going to use the bus. What is bus? Bus is a set of connecting wires and the beauty of this is that everybody sees the bus. Right. For snooping we need medium where everybody can check what is happening.

So bus is the good medium for this. Okay. So whenever a request goes on to the bus every block will be able to see this. Right. So when I am changing u equal to 7 somehow I broadcast this. I tell everybody so I shout it out here that saying that u is equal to 7 is happening. So this goes over this bus and each of this can see that this is happening. Okay.

So every cache controller can see what is happening on the bus. So P3 if P3 tells that u equal to 7 is going to happen everybody can see that yes u is becoming 7 and they can take appropriate actions to update their local information. So the key property of bus snooping is that every transaction appears on the bus. They are visible to all controllers in the same order. So this is also important that if there are two updates happening they will happen one after the other.

So if u equal to 7 is done by P3 and after a while P4 does u equal to 8 then first this is going to travel on to the bus and after this then this will travel on to the bus. Okay. So the order is also maintained by this broadcast medium called the bus. So everybody will see this in the same order. And once they see this what should they do? They have to take appropriate actions. So in this when P1 and P2 will see u equal to 7 what should they do? Well it depends on the protocol but if you simplistically say that yes when u equal to 7 is going on to the bus let P1 and P2 update their local copies. Right. So they will simply say okay u equal to 7 let me just take it up and write it into my cache and if I do this both of them will get the most up to date answer. Okay.

So that is the basic idea of how snooping would work. So for doing this implementation what are my system requirements? My system requirement is that first I need a broadcast medium. In this case I am assuming the bus. So when I have a bus it should follow some protocols. Right. So bus transaction also has a particular order in which it works.

So, if you recollect from another basic course that how does one send a request onto the bus? It goes through three phases. The first phase is called arbitration, second is called command and address and third is called data transfer. So what is arbitration? Whenever multiple devices which are connected to the bus want to access or send something on the bus they cannot just start sending. They need to take permission before sending because the bus being a common resource you need to first take permission before using it. So you take the permission from a bus arbiter.

So first every requester has to go to the bus arbiter and then ask for permission. One of these devices will get permission to send. So once you get the permission or the grant, after the grant signal comes then this particular sender has to put the command that is what is the request it wants to handle and the address. So in my case it could be a read or a write. So we will say that I want to read or I want to write and then we also have to send the address that this is the address to which I want to perform this action.

So this is a bunch of wires, bunch of wires carrying the read write signal, another bunch carrying the address and when this command and address is put, the third phase is of data transfer. So if I am going to do a read then some sender has to give me the data. So assuming that the memory is going to send the data, so once the request of read travels on the bus, the main memory will see this and put the data onto the bus for me to read. Okay. So that is the data transfer phase when we are doing a read. If we are doing a write then we are going to send the write signal, the address and then we put the data, so the receiver that is in this case probably the memory is going to pick up this data from the bus and update itself. Okay.

So that is about bus transaction and for implementing the coherence we need another thing called the state or the state information about this cache block. And when I say state of a block, the state of the block could be variety of things and we have at least two or three states present but if you make a complicated protocol you might have several states. So I actually need a state transition diagram to do this. Okay. So we need a state transition diagram with every cache block. Till now you know that a cache block consists of a tag and data and it also has these three states which are popular to us whether it is a valid or invalid block and whether it is dirty. So we have seen this in the recap about how caches are designed.

But apart from this I want a state transition diagram which is nothing but a finite state machine which shows how the cache is changing the state. So it is nothing but an FSM which will tell how the block transitions from one state to the other. Suppose initially the block is not present so the block is invalid. Once you do a read for the block, the

block will start becoming valid because the V bit will become 1. If you do a write to this what will happen? The dirty bit will become 1. Okay.

So this is a sample of how an FSM can be constructed. So the system requirements for coherences we need good bus transaction support and we need a state transition diagram for implementation. So let us look at the concept of FSM of a cache block. So I am assuming this as a cache. So this P1 means this cache is belonging to processor P1 and blue is some block, some address and pink represents another address.

So these are the two cache blocks. Now for this blue block I have a finite state machine associated with it. Suppose it was invalid then it becomes valid then it goes to some state as 0 then to S1 and so on. So I have just drawn four sample states but it could be any elaborate design here. For the pink one also I have a parallel FSM design. So every block has got a finite state machine associated with it.

Now these finite state machines are present or handled by the cache controller. So how does the system look like? We have the processor. The processor connects to the cache. Cache has got the cache controller and this cache controller is managing all these finite state machines. Eventually it will speak with the bus through the bus snooper.

We are discussing snooping protocols. So we need a controller sitting at the bus which sees what is happening over the bus. So the bus snooper is going to see what is happening and then take appropriate action. Okay. So when the processor sends a request for read or write it goes to the cache controller. The FSM may be updated for this. Right. It may update the FSM and then send, this arrow should be like this, it sends a request over the bus snooper then the bus will send a transaction.

So a transaction happens over the bus. Once this transaction completes we will get a response and eventually that response will get transferred to the processor. So this happens for every cache block and therefore we have a finite state machine associated with every cache block. So this blue is the FSM for this block for suppose this is some address B and this is some address. Okay. Block B0, block B1 they are two different blocks and they are the two different FSMs for this particular block. Okay. So if you now can understand or imagine that the cache has got so many blocks and there is an FSM associated with every block. Right.

So in a multiprocessor there are multiple caches, every cache has got multiple blocks and every block has got an FSM. So you can imagine the scale of the problem. So we will first illustrate and then come back to this slide. So here I have drawn three processors P1, P2 and P3. Okay. Each has got presently only two blocks just to take a

small example.

Processor P1 has got a blue block and a pink block. Processor P2 has got a green block and the blue block. So you can say that the blue block is common between the two. So it is a shared variable. Okay. Similarly P3 is caching the pink and the green block.

So the green block is shared with P2 and the pink is shared with P1. Right. So if you start drawing the FSM, so you have first the FSM for the blue with respect to P1. Then you have the FSM for pink with respect to P1. Okay. So I will go back here.

So this is the FSM for the P1 processor. This is the FSM at the P1 processor. Continuing further, this is the FSM for the P2 processor's, blue block. Then P2 processor's, green block. P3's pink and the P3's green. So in this example, we have these FSMs implemented and you will also see that processor P3, it does not have the blue. Right?

The blue block is not present here. So what do we do? What is the state of the blue block inside processor P3? If I ask you this question, what is the answer? That P3 does not have the block and when I say it does not have the block means the block is invalid. Okay. So that is why I will have another FSM associated with P3. Right. So this one where, so some FSM for P3 for the blue block which says that it is invalid. So essentially every process will have the FSMs for all the blocks. Right.

So for P2, the pink is absent. So I will have the pink as an invalid and then of course it will have the other states which I have not shown in the diagram. For P1, the green says that it is invalid and I will have subsequent FSM states for this. And why do I need this? Theoretically I need them because whenever P1, suppose it brings the green block, it should be able to handle the state transition. If it does not have an FSM, it cannot handle it. So we have to theoretically have a FSM associated with this. Okay.

So now that we have seen the FSMs associated with every block, let us take the big picture of having a multiprocessor where every block in every cache has its own FSM. So what is the global state of the caches and the shared blocks? If I have P caches in the system, then every cache will have some state of this block. So if I am talking of a block, say the blue colored block, what is its state in these three processors? I can say that it is invalid in something, it is valid here or modified there. So essentially I have actually three values.

One value per processor. So I can say I have a vector of size p. Okay. So in a multiprocessor system for each block which is sitting across P caches, I have a vector of

size p which is telling the state of the block. Okay. So state of this block can be kept in a vector of size p. In my example, the vector was of size three because I had three processors. So I will say this block, how is it in these three processors? It could be invalid here, it could be valid here, it could be valid here.

So what is the state of the block in each processor is kept in a vector. And the manipulation, that is the state transition is happening across P distributed FSMs. If you go back to this figure, if I am discussing about the pink block, then I have a FSM in P1, another FSM in P3 and probably one more FSM in P2. So there are three FSMs for the pink block. Right. So all these FSMs are going to run in parallel and they are going to coordinate with each other to maintain the global state. Okay.

So same example, here I have drawn FSMs for the pink, sorry for the blue block. So this is FSM at P1, FSM at P2 and P3. And if I want to understand the global state of the blue block, what will I do? I will construct a state vector and this state vector will say that the block is valid here maybe in some other state here and invalid in P3 and so on. Okay. So this state vector is going to tell what is the state of the block in the corresponding processor. And when any change happens, suppose this moves from here to here, we are going to broadcast this information over a bus and looking at this change probably P1 might take some action. Let us say that here this one starts writing, P2 starts writing to the blue block and it puts that information on the bus.

So what should P1 do? P1 will say hey he is writing it, so either I update my information or I invalidate my block. So this is the action taken by P1 looking at what is happening in the other processor. Right. So the snooping protocol is working as nothing but a distributed finite state machine. It is a collection of FSM's which are coordinating with each other. How do they coordinate? By sending information over the broadcast medium that is if processor P1 does any change in the blue block, it is going to put it on the bus, others are going to see what is happening and make changes in their local FSM's.

So it is coordinated by the bus transactions. So I hope it is clear to you that every block has a local FSM and any change in one FSM state is sent as a bus transaction and other processors update their states. Right. So essentially what do we have? We have states of the memory blocks in the local caches, a state transition diagram associated with every block in every processor and then actions associated with it. That is what action to take when a particular transaction goes over the bus. Okay. Snooping protocol is nothing but a distributed finite state machine which will coordinate with each other to maintain coherence for every shared block. So we have a coordination among the pink FSM's, others with the green FSM's and so on. Okay.

So we will discuss more details about this in the next lecture. Thank you so much.