Parallel Computer Architecture
Prof. Hemangee K. Kapoor
Department of Computer Science and Engineering
Indian Institute of Technology Guwahati
Week - 05
Lecture - 29

Lec 29: Coherence misses example

Hello everyone. We are doing module 3 on cache coherence. This is lecture number 11 and in this lecture we are going to see a detailed example of the cache coherence misses and along with some classifications in related to update versus invalidation based protocols. So we will start with the example of cache coherence. We have seen the different categories of misses and now we will do an example, okay, to understand the process properly. In this you can see a table on the slide and we are going to fill it slowly.

Here we are going to list the time sequence that is the sequence in which the events are happening and then what happens in every processor. We have three processors P1, P2 and P3 and in the last column we are going to fill the miss classification name. We are only going to discuss related to one block. So we are going to take one block B and this block has got, as got four words W1 to W4.

So there are four words sitting in this cache block and other words, words 5 to 8 will also sit in the same location so that we can generate the conflict scenario. Okay. Then I am also going to use some new notation which says that what the processor ID along with the time stamp. Okay. So looking here if you see I am going to use one cache block having four words and another cache block again having four words and both of these map to the same location in the cache. So they map to the same location meaning they will evict each other. Okay. And then this notation P(i,j) it says memory reference is issued by processor i at row j.

So row j is the time stamp. So this is the time stamp and that i is the processor ID. So if I say P(1,3) what does this mean? Processor P1 and time 3, time 3 which is row 3. So it is talking of this cell that is what happened in this cell. Okay. So we will start with the example.

Here I have filled all the events in the first three columns and we are going to fill the miss classification. So I would say that you try pause the video intermittently try to solve it and then restart the video. So we look at the first row time 1. Here processor P1 and P3 read W1, W2. Okay. And initially there is cache is empty.

So this is the first access and it is the first access and what happens here P1 and P3 will incur a cache miss. They will incur a cache miss. What type of miss is this among the 12 names which we looked at? I do not know right now because we will only categorize it when this particular block gets evicted from P1 and P2. Okay. So when I remove this block I will give some name to this cache miss. So I want name I will simply say it missed but reason is not written.

Row number 2 here. P3 writes W2. So what will you do here? This write is going to invalidate this one. It is an invalidation based protocol so it will remove that and when P1's block gets removed I can now give a name to its miss. So I will say P(1,1)means processor P1 time 1. What happened in processor P1 time 1 that is what happened in this particular cell?

What was the reason for this cache miss? I can tell this reason after it gets invalidated. So we can say it was a pure miss and cold miss because W1, P1 brought and it used and nobody has changed W1 after that. So I will say this was a pure cold miss. Okay. So I hope you understood the notation that whenever the block gets evicted from a processor we are going to do the miss classification. So P(1,1) pure cold which I wrote here I wrote it in row number 2 because in row 2, P3 evicted P1's block and same with P3.

What happened in P3? Can P3 directly write? No it cannot. In any coherence protocol P3 has to first get permission to write and hence it has to send a bus update or a bus inval and so on. So P3 and timestamp 2, I will say it incurred something equivalent to an upgrade miss. P3 had to upgrade its block for writing. Okay. Third row P2 wants to write.

So I will say simply P2 will incur a cache miss. We cannot categorize it right now. We categorize it later. Third row done. Fourth row. Fourth row this will it hit or miss? It hits right because word 1, 2, 3, 4 are in the same block.

So because of this there is a cache hit. What about this? Read W7 by P3. So here P3 is it going to hit or miss? It is going to miss because W7 is in another block. The word 7 is in the second block and not the first block.

So P3 will miss. I cannot categorize this miss right now but because of this miss which block gets replaced? The first block. Okay. So if I say B1 and B2. So let me say B1 has the words W1 to W4 and B2 has the words W5 to W8. Okay. So if I use this notation here this is block B2 and this block B2 is going to evict or replace block B1. Okay. So they are different blocks and because I evict block B1 from P3 I can give a name to it P3

and timestamp 1 because the block was brought in the first timestamp.

This I will call as a cold miss. When the block gets evicted we give a name to that. Okay. Row number 5. Is it a hit or a miss? It is a miss because W5 is in another block and it has block B1 but B2 has W5. So P1 misses nothing else happens in this row or this timestamp.

Moving on row number 6. Will this hit or a miss? It is W6 block B2. So I will say P2 also incurs a cache miss and now here we need to categorize the other misses that is when I read this I am going to evict the block B1. So let me write this first.

P(2,3). P2 had loaded block B1 in timestamp 3. Okay. Right. This has led to eviction. This has evicted that block and we can now categorize the reason that is why did we remove this block meaning what was the reason for miss at row number 3. So what was the reason for miss? It was definitely a cold miss but was it a true or false sharing miss? I am talking of this block. This particular block was there brought for the first time. So I am saying it is a cold miss.

But if you see it is accessing the word W2 which was modified by P3. Okay. So even if P2 would not have incurred the miss it would have resulted into an invalidation by P3. So it is using W2 which was changed by P3. Hence I am going to call it a true sharing miss. True sharing but with an additional clause of cold.

So we have done this. Now if you see let us revisit row 5 because row 5 also evicts this one. Okay. So if I categorize P(1,1). P1 row 1. What happened in P1 row 1? We had a cache miss. It was definitely a cold miss.

But was it a sharing miss? What did we access? We accessed W1 and W1 was not changed by anybody in the system. Hence we will call this a pure cold miss or just a cold miss. Now row number 7. Row 7 write W6. Is it a hit or a miss? Row 7. I would say P2 has the block already but that block is also shared by P3. Right.

W6 is block B2. Block B2 is present in P3 as well as P2 is also trying to access it. Okay. So row number 7 there is a write for word 6. When P2 writes word 6, it hits in P2 but I would say P2 was reading it and now it wants to write. So we can, let us quickly say that it is an upgrade miss for P2. Right.

So let us do the easy case. For P2 and in row number 7, the current row I can say this was an upgrade event for P2. But because of this we are going to evict this block and we are going to evict this block or I would say invalidate because we are modifying it. The

block gets invalidated from P1 and P3. Hence I need to categorize them P1 and P3. So P1 had loaded this block in row number 5 and P3 had loaded the block B2 in row number 4. Okay.

So what are the reasons for this? So P(1,5), P1 in row 5 we loaded the block and what is the category of that miss? It is a cold miss because it is accessed for the first time. And W5 was not changed by anybody so I can simply say this is a cold miss. And P(3, 4), P(3 ,4) is this one and we are going to, what is the reason for this block? We are reading W7, did anybody else change W7? Nobody changed. Right. So I can also call this a cold miss and also we can put the word pure because it had to happen and it is not related to coherence. Okay. So with this, this was the explanation but a neater final answer is on the next slide. And we will have here one P, so these are 7 time stamps, we will do few more.

I hope we will have to recollect or refer to the previous slide because we need to remember what happened previously. So looking at row 8. Okay. Because it is a continuous example, row 8 we have a read W5, does P1 have W5? If I go back here, because of this, this was evicted, hence P1 does not have W5, so I will say at row 8 P1 misses, there is a miss in P1. Row number 9, is it a hit? Yes it is a hit, so I will say P1 does a hit. And what about P3? P3 does a load that is a read. Okay. So this is a read W2, if it reads W2, does it have W2? I will go back to the previous slide and you can check.

P3 recently had 7 but it also got removed, 2 is also not there. Okay. So P3 also miss. I will request you pause the video intermittently to cross check and then try to solve it yourself. Okay. So if you do it meticulously, slowly, you will be able to do this correctly.

So we are at row 9 done. Now row 10, row 10, read W2, hit or a miss? It is a miss because block W6 is another block and what about this, read W1? Let us go to the previous slide and see P2, P2 presently has W6 that is another block, 1 is not there. Okay, so both of them miss. I will say P1 miss, P2 again miss, row 10. Now because we miss in row 10, in P1 we are going to evict, we are going to delete this block. Now what is the reason for bringing this block in row number 8? So I will say P(1, 8), P1 brought the block in row number 8 which I removed just now.

What was the reason or what is the label for that miss? The label is we used 5 and 6, right, we removed, just now we are removing W5, W6 but were W5 and W6 changed in the meanwhile. So if I go back to the previous slide you can see, yes the W6 was changed by P2. So this W6 which I read was changed by P2. So P2 gave me the best answer or the most recent answer and hence I will say it was a sharing type of a miss and it is a true sharing miss. So I can say this is a pure true sharing miss because I will just

say W6 for your reference that I access W6 and that is why sharing miss.

What about P(2,6)? I will write it here, P(2,6). P2 has to, when it reads this, go back to the previous slide you know that it is going to evict this block. This block will be evicted and what is the reason for bringing this block? That is the reason I am going to write in the next slide, P(2,6). P(2,6), what is the reason? It was a pure cold miss, there was because it changed the data and did not use anybody else's data. P2 did the writing to W6, it did a read on W6, nobody had changed W6 apart from P2, hence it is a pure miss for, cold miss for P2 and not a sharing miss. Row 11, is it a hit or a miss for P1? It is a miss because W5 is in the other block, so it is going to remove the block of row 10.

Okay, so when I remove a block P(1,10), what should I write here? P1 had brought a block in row 10, I am removing it right now, so this block only stayed for one time stamp. What type of a miss shall I call it? Is it a sharing miss? So for that you have to see the word W2 and was W2 changed by anybody in the system? On this slide definitely not, but if I go to the previous slide, here you will see at time stamp 2, P3 had changed W2. Okay. And P3 changed W2 and P1 reads the W2 in time stamp 10 Hence I will say this is a true sharing miss. So in row number 11, P(1,10), I will say true sharing miss with reference to word W2.

So far okay. Now row number 12, write W2 happens, is it a hit or a miss in P3? Row 12, the block is there, yes it is there, but I will say this is an upgrade because P3 in time stamp 12 is an upgrade. And when P3 accesses W2, what does it do? This access is going to evict this one. It is going to invalidate the W1 in P2 and when I remove that I need to make an entry, what happens to P2 for the block it had brought in time stamp 10. Okay, so it access W1 and nobody changed W1, so it is not of true or false sharing and it simply happened because of capacity and not a cold miss because I have access W1 earlier in time stamp 3, then I had to remove it because I read W6 and I had to reload it again here. Right. So this is happening because of capacity. This is not a cold miss, cold miss for W1 has already occurred here. Right.

This is already a cold miss, but the second access to W1 is a capacity miss. Okay, row number 13 we are seeing P3 incurs a cache miss, it is going to evict this block. So I am going to write here P3 has a miss, it evicts the block of P2, this particular block. So I need to give a reason P2, but when was this block in row number 12 loaded? It was loaded in row number 9, right. So P3 row 9, P3 row 9 because the block with word W2 was loaded in time stamp 9 and we changed W2, we read W2, but did anybody change W2? On this slide nobody changed, previous slide. Only P3 changed W2, nobody else changed. So we are only changing, nobody is changing that, so I can say that this is a capacity miss, it is not a coherence miss, it is a capacity related miss.

So we finished this case of row number 13. Now let us again have a quick look at P1, when we do read of W7 here, there, this block is also present in P1, so do we evict this block? No, we do not, only thing will that this block if it was in modified state, it will become a shared state. Okay, so that will be information transfer between P1 and P3 for this case. So it will move to the shared state. Okay. Moving to row number 14, row number 14 is read W2, again it is a miss, so I will say P3 is a cache miss because this block has to be removed now, so P3 block W7 in row number 13, P(3,13), what is it now? Pause the video and give me the answer. Processor three row number 13, that block what category miss should you call it? Okay, so is it W, we access W7, did anybody change W7? No, so it is a false sharing. Okay, but that block if we had not removed it because of capacity, if you see here, use another color, see because of this relationship, the block would have been invalidated. Okay.

Even if we would have not removed it, but it would have been invalidated in some sequence and hence I will say it would have been invalidated. Okay. So it would have been an invalidate related miss. Presently it is also a capacity miss because of this too, so it happens because of the capacity we had to remove and reload it and it is a false sharing because we are reading W7 and P1 has changed W5. Okay. So it is an inval capacity false sharing miss, okay, that is row 14. Okay, we will do row number 15 now, the last case, read W1, P1 hit or miss?, miss and it removes the block from row number 11. So P1 row 11, what happened there? 11, we had brought the word W5 that is the second block and we brought that and now we have to remove it. So nobody else wants this block, but we are removing it and when we brought this block at 11, we had to remove this block.

So between these two, if you see row 10 and 11, this was a capacity issue and not a coherence issue and hence I will call this as a capacity miss. P1 and 11 is a capacity miss. Right. So this is the solution and a neater solution is given here for your reference. Here I have also written when I say true sharing miss in say row number 11, which is the word which we had accessed and so on. Okay. On this slide, although it is very tiny font written here, but I have written the full example but definitely the pieces of this are in the previous two, three slides. Okay.

So for your reference, in one chart, the whole example is written. Right, so that was the misclassification example. Now we are going to look at methods of reducing coherence misses. Well, the basic idea or the best solution for reducing any type of misses increasing the cache block. Right. So we are going to increase the cache block, but they come with some drawbacks.

But why do we have large cache blocks? Because we have a increasing gap between the processor performance and the memory bandwidth. So memory access time is slow, processors are fast. So when we go to the slow memory, we want to bring lots of data with us. So when we bring lots of data, it will help us in temporal spatial locality and improve the performance.

So we want large cache blocks. Okay. And these large cache blocks create problems for multiprocessors because of this sharing problem which we are discussing right now. The larger the cache block, more the false sharing misses which happen. So the gap between the processor and the memory is helping us or telling us that I should bring large amount of data and hence have larger cache blocks. So the trend of larger cache blocks has been coming because I have multiple high density processors.

Then I can also have multi-level caches. I can do prefetching of blocks. So all these high end processors, several levels of cache hierarchy and also prefetching data blocks. So prefetching means go and bring the data from the memory before it is required. So when I go to bring something beforehand, I would rather bring more stuff than less stuff. Right.

So prefetching would be benefited if my cache blocks are large. Okay. So these are the reasons why I have a large cache block but these have problems in multiprocessor systems because they are going to increase the false sharing misses. False sharing miss increases so you have more invalidations and unnecessary bus traffic occurs. Okay. So how do I counter the effect of these large cache blocks? Okay. They are affecting my false sharing misses. So we can say tha,. Okay. so false sharing misses increase because of a large cache block. How do I solve this? So you will say that we will only concentrate on the false sharing aspect that all the blocks which have some words which are not used by everybody, let them not sit in the same block. Okay.

So what I will do, I will organize the data such that only the words which are used by everybody will sit in one big block and those sparingly used words by others will not sit in this block. So even if the block size is big, my false sharing will be less. Okay. So we have to organize the data structures. Then we will say that one solution for this is can I have the coherence maintained at a smaller granularity.

Right now coherence is at the granularity of the cache block. So if, even if I am accessing one word, I end up invalidating the block across all processes. So if I say that coherence is only word level, so the word which I am changing, only that word across all the processes will get invalidated whereas not the whole block. So the block can stay but the words will get invalidated. So that is one solution. And if you do this, then the

updates or the changes to be done to the memory, the traffic related to the memory or the interconnect traffic also reduces because you are only going to change the modified words and not the whole block will go to and fro between modules. Okay.

And if you have smaller cache blocks, you will say it is going to take more time to load them but we can say yes, you can always use prefetched data to cater to these small blocks. Okay. So countering the effects of the cache block, I need to organize my data structure such that I have more properly or more shared data sitting in the same block. Right. So this is handled mainly in the software. So we can say somehow the programmer manages to pack shared items into closely packed locations and less shared items in different locations. So I am going to fetch at the granularity of the cache block but coherence in small sub blocks.

So how is this? So look here. Suppose this is the green is one block and the pink is another block. So this whole thing is one cache block but instead of invalidating or so instead of removing this whole block at a time or whatever, I am going to control it at the level of the words or sub block. So this is one sub block. So when I change this particular word, we are only going to invalidate this word across the system and not the others. Okay.

So certain blocks could be valid and certain blocks could be invalid in this. Why invalid? Although the data is there but I will say this is being modified by some other processor. This sub block is being changed by another processor and this sub block is changed by this current processor. So within the block we will make partitions and say that this partition I am modifying, that partition somebody else modifies and so on. Okay. So this is also reducing the amount of data written because only this much data will be sent on the bus by us because we only change that particular word. Okay. And small caches, cache blocks are okay because we can always prefetch the data beyond the block size.

So this is how I can counter the side effects of having a larger block. One is by organizing the data and packing more access to data items into a large block, one option. And second is another one where I maintain coherence at a smaller granularity of the sub block level. Third solution is use adjustable block size. That is depending on the type of shared data items you adjust the size but this is a common complex solution. And the fourth option is this problem occurs because we end up invalidating cache blocks. Right.

So if I have a big cache block and we start writing it we have to invalidate it across all processors. Later those processors incur a miss, they will again fetch this block. So there is lot of traffic happening. So we can say that let us consolidate all these writes together

and then eventually send the update or invalidations.

That is only when I change a single word do not do anything. When few words change in your block only then you inform the system that invalidate your block and that I have fresh data with me. But what is happening here I am accumulating lot of writes and then sending it across the system. This might affect the memory consistency models which we will discuss later. Okay. So delaying the propagation of invalidations is okay but it affects the memory consistency models. So what is the fifth solution? Fifth solution is use a hybrid of update and invalidation or just use update instead of invalidation.

Because the problem comes because I am invalidating do not invalidate, just keep on sending the updates. Okay. So with respect to the fourth solution this is a example to delay the propagating of invalidates. Here if I change this I have to send invalidation but do not send it right now, wait for a while, when all three have finished then you invalidate. Once you have finished all three invalidate or send the updates on the bus.

So we can send all of them together but this solution affects the consistency models. Right. So to counter the effect of large cache block, we said that we can use update protocol instead of invalidation or some variations of this. So let us see or understand when I use update protocol and when I should use an invalidation based protocol. So this actually depends on the type of sharing or the workload patterns which we are running. If there is lot of sharing between the data items then we need to decide whether updates would be good or invalidations would be good. Okay. So when is update good? It is good when the updates which I am sending on to the bus, the receiving processors keep using those updates, that is all the updates are consumed and locally utilized.

In that case update is good. But invalidation is good in the opposite case, that is this processor changes data and nobody else wants this data right now. Right. In future it might need but right now it does not need it immediately. So even if I invalidate those blocks from those processors it is not going to hamper the performance or increase the traffic. Okay. And invalidations are also good in the cases when I have a single threaded application, it is running on this processor.

After a while operating system reschedules this process on another processor. So it was running on processor P1 now it runs on P2. So when this process moves to P2 and run starts running there it is going to incur cache misses in that processor whereas its stale data is still in this processorm in a valid state. So when it runs on the other processor it is going to keep on sending updates and invalidations to its stale data in the previous processor even if it is not using it. So that is unnecessary traffic and in this scenario if I use invalidation based protocol then the staled cached copies in the previous processor

will get removed, right.

So update versus invalidation depends on the sharing patterns. I use update when the updated data is likely to be used by the new processor, right. So it keeps using it so good for us and I will keep sending updates and invalidation based protocols are good when the processors are never going to use those data items. So it is going to reduce your useless traffic happening. I explained that there is a process running on P1, OS reschedules it to another processor, but the stale data remains on this and new accesses keep happening here right. So the process itself has moved to another processor and its staled copies are in the previous processor's cache and these will keep on getting updated in the case of update based protocols. But if I invalidate it then these copies will get removed, okay.

So unnecessarily lying copies across the system, useless copies of data will get removed in case of invalidation based protocol. Can I use a hybrid of update and invalidation? Okay. So this is a good idea but who will do this? We have either hardware for update or we have either hardware for invalidation. So we can say that can we move the burden to the programmer and say that the programmer manages the locality or loads the variables in such a way that more shared variables that is one type of access variables sit at one page. That is at page granularity, I will say that this particular page, this is not a cache block, memory pages. So page 1, page 2, this page uses update protocol, this page uses invalidation based protocol. Okay.

So can I do this? So we leave the burden to the programmer that the programmer says or manages the variables such that a particular type of variable which suits update protocols sits in one page and the types of shared variables which suit invalidation based protocols sit in another page, right. So we have two different pages with categorized variables kept in them correctly. But this puts a burden on the programmer to do this and to implement this we can make note at the TLB level. So TLB says that this particular page uses so and so protocol, okay. So that could be done but can we do it in hardware? Okay. So in hardware if we want to do, one solution is you can say one type of a cache uses invalidation, say L1 cache uses invalidation, L2 cache uses update or we can change dynamically.

That is I am running a protocol of update for example and I keep on sending updates on the system. Now these updates get consumed by other processors who have the cache blocked. But if those processors are not using those cache blocks, then if somehow there is a mechanism when they will say that hey you are sending me updates but I do not need them anymore. Let me delete the block myself because you are sending me unnecessary information. I will delete this so that you need not send that information to

me, okay. So if we can implement such an idea then at runtime we can switch between update and invalidation. So we will see how could that be done, okay. So to manage it in hardware we could either say that I have two caches working at different methods. So L1 works on update, L2 could be based on update and L1 based on invalidations or the second one is we start with an update based protocol and then at runtime change to invalidation or vice versa.

So next slide has the example. So when I talk update versus invalidate I am going to refer to the Dragon protocol. Here I will have a block B and I will set a counter K to it, okay. So whenever an update comes to this protocol right, through the dragon protocol when a bus update is coming I am going to decrement the counter because I am going to keep track that how many times am I getting updated without getting used okay. But in case B gets used in this processor, if I use it then when I use the block the updates were useful for me right.

So whatever data come was useful for me because I am using that data so I will reset the counter. So I will do K. Suppose your K was equal to 4 and we did not access the block and update came we go to 2, 3 and suddenly we start accessing, so yes the updates are important for me so I will reset the counter to K, okay. But if the updates are not important for me I will keep on decrementing the counter. So every time an update comes we are going to do counter minus minus and if K becomes 0 what will we conclude? We will conclude that all the updates coming to us were not important. We are not using them so we might as well delete the block from the system. Okay. So we will locally invalidate this particular block and when we invalidate this block what would happen? The future bus updates will go and the shared wired-OR signal will say that this block is shared by nobody and so we will switch to modified state instead of update.

So we will not send bus updates anymore, right. So here if the counter reaches to 0 the block gets deleted and the sender that is the owner of the data who was sending us the bus updates will now switch its state in the Dragon protocol to the M state and in the M state no bus transactions happen. So what have we done? We have reduced the bus transactions. Essentially not directly invalidations but the bus updates have been removed because the block itself invalidated itself. Okay. So now we will switch to the block locally invalidated without the protocol actually changing to invalidation based protocol. And as I discussed earlier, if we access the block then you reset the counter to K because I am accessing the block the updates are important to me and hence I should not remove the block. And when we have the block it will resume the bus updates. So now we will see that the bus updates are important reset the counter and keep receiving the bus updates, okay. So the overall idea is, if the bus updates are important, keep on receiving them, if they are not important, invalidate the block in the local processor,

okay.

So with this we have understood the variety of scenarios that is when update is good, when invalidation is good, can we switch between the two, what are the different methods and so on. And in this lecture we have also seen the misclassification. Thank you so much.